Estimating Software Development Task Effort Using Word Embeddings and Recurrent Neural Networks

Mariss Tubelis

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Anders Kofod-Petersen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

This cross-discipline project tests a state-of-the-art neural network model on a problem with high impact in software engineering, namely task estimation. The majority of research conducted in the software estimation field focuses on early, prior-project estimation, while considerably less effort has been spent on task estimation, which has become more important since agile practices became widely adopted.

In this paper a dataset consisting of more than 63,000 tasks with textual descriptions and time spent reported was created from 32 publicly available JIRA issue tracking system instances. Five architectures of LSTM and highway neural networks were then parameter-tuned on 19 subsets of the main dataset by running 18,000 evaluation rounds in total. The use of general English word embeddings was compared with learning word embeddings from more than 2,000,000 publicly available software task description text corpus. The results were validated on two commercial datasets of 9,000 and 30,000 labeled datapoints respectively.

Although the results were not gratifying as the model accuracy wasn't anywhere close to human expert accuracy, this project provides a solid contribution to further research in the field by describing the methods applied in the attempt to solve the problem as well as several observations regarding transfer learning effects and optimal model configurations. The main dataset and the results together with well-documented data gathering, preprocessing, model training and visualization scripts were published on GitHub[1].

[1] https://github.com/marisst/Bestimate

Preface

As a software engineering student with industry experience both as a developer and project manager, I have always wondered if the estimating process could be made more data intensive as opposed to the often biased, but currently more accurate human judgement. Although artificial intelligence is not my primary study field, I had the interest to explore the capabilities of neural networks. The task at hand is extremely difficult even when estimating uncertainty is perceived in a healthy way and I hope that others with more experience in artificial intelligence will will continue the research.

I could not have made this project without the support of these people and their represented companies:

- **Anders-Kofod Petersen** for supervising the project
- **Arturs Kruze**, CEO at Magebit, and **Kristaps Rjabovs**, Managing Director at Magebit, for allowing me to validate the model in their company
- **Antons Sapriko**, CEO at Scandiweb, and **Davis Krikauskis**, Executive Partner at Scandiweb, for allowing me to validate the model in their company
- **Martin Baklid**, for advisory on neural network model development
- **Patrick Tanner Coursey** for proofreading the text
- **Ints Skaldis**, for legal advisory on confidentiality agreement
- **Krista Lubina**, for advisory on communication
- **Runar Ovesen Hjerpbakk**, Software Engineering Manager and Head of Trondheim office at DIPS ASA, for ideas regarding collaboration with the companies
- **Joakim Granli Antonsen** and Telenor AI Lab at NTNU for providing access to their server cluster and for technical support

I also want to thank these people and their represented companies for their time and an honest feedback on my project:

- **Maksims Jegorovs**, Accenture Latvia Lead, and **Kristaps Banga**, Innovation Lead and New Business Developer at Accenture Latvia
- **Iveta Bukane**, Managing Director at Visma Labs

Contents

List of Figures

List of Tables

Glossary and Abbreviations

Data label	Time spent on execution of a finished programming task
Datapoint	Textual description of a programming task
Labeled datapoint	Textual description and time spent on a finished programming task
Labeling coverage	Ratio of labeled datapoint count to total datapoint count
Mean estimate	Estimating future task effort as mean of time spent on past tasks
Median estimate	Estimating future task effort as median of time spent on past tasks
Project size	The number of finished programming tasks with time spent reported in a project

API	Application Programming Interface
CBOW	Continuous Bag-Of-Words
CSV	Comma-Separated Values
JQL	Jira Query Language
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
MSE	Mean Square Error
NDA	Non-Disclosure Agreement
NLP	Natural Language Processing
NN	Neural Networks
OS	Operating System
REST	Representational State Transfer
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Networks
SGD	Stochastic Gradient Descent
TPE	Tree-structured Parzen Estimators
URL	Uniform Resource Locator

Chapter 1

Introduction

In this chapter the reader is introduced to the background of the research, the research goal, the research questions and the motivation behind them. Next, the method used to achieve the goal is described. Further, the contributions of this paper to the research field are summarized. In the end of this chapter, an overview of the thesis structure is given.

1.1 Background and Motivation

So far, most researchers in the software development effort estimation field have been focusing on estimating whole projects prior to their start in order to determine the total budget and time needed to develop information systems. Many different techniques have been tested to create estimation models that could predict the cost of a software system given a set of parameters indicating the complexity of the project and the productivity of the developers (Tubelis, 2017). However, such models aren't widely adopted in the industry because they are too complicated to configure and use and, although often very biased, human judgement based estimates are still more accurate. Moreover, agile principles have become widely adopted during the recent years and in many companies the software development life cycle is product based and incremental. The focus of software estimation is therefore shifted from early estimates of the whole project to late estimates of programming tasks.

Most companies use an issue tracking system to manage programming tasks and in many companies employees report the time they spend on the tasks in the same system. The accumulated data about actual time spent on task estimation can be used as a basis for estimation model development. This opens an opportunity to close the gap in the research on programming task estimation. Moreover, if an accurate model will be developed, the chances of its adoption are very high, because most of the popular issue-tracking systems are built as platforms allowing for external innovation in the form of plugins. In 2017 two such plugins were developed, but they didn't become very popular, which is most likely due to low prediction accuracy. Similar solutions have been successfully adopted in other domains, such as user support task estimation. However, because of the heterogeneity and the noise in software development descriptions and time spent, such models are yet to develop in software development task effort estimation domain.

1.2 Goals and Research Questions

Goal Construct a neural network model which estimates software development task effort from textual descriptions in English significantly better than mean and median baselines given the actual time spent on task execution.

The core component of a software estimation plugin in open project platforms would be the actual machine learning model making the estimates. Developing a good interface with the platform would not provide any value without the actual estimation model. Therefore the author decided to start by developing the minimum viable product, which only includes the most essential parts of the solution, namely the estimation model.

Research Question 1 How much labeled data is publicly available, and how much is necessary to achieve the Goal?

It is well-known that machine learning models need substantial amounts of data to learn complex relationships. So far only one paper has been published investigating publicly available data from project planning platform instances using up to 23,000 datapoints (Choetkiertikul et al., 2018). However, Choetkiertikul et al. (2018) have not particularly focused on reported time spent on task execution. It is also important that the data on which the model trained is publicly available so that further contributors in this field can easily reuse the data and reproduce and improve the results.

Research Question 2 What are the noise and heterogeneity levels in publicly available software development task descriptions and time reports, and how can they be decreased?

Noise in input data can be a disturbing factor for machine learning models. It is therefore important to reduce the noise by preprocessing the data when possible with structured approaches. Such approaches can also be reused in further research focusing on natural language processing in the software engineering domain.

Research Question 3 Does learning word embeddings from unlabeled datapoints have an advantage over using publicly available word embeddings pretrained on general English text corpus?

Since neural network models operate with numeric weights, textual task descriptions written in natural language need to be converted to numeric representations. In most cases it is done by learning word vector embeddings on a text corpus. It needs to be clarified if utilizing unlabeled task descriptions, of which there is plenty in public repositories, for learning vector embeddings have an advantage as opposed to using off-the-shelf word embeddings trained on general English texts.

Research Question 4 Which neural network configurations provide better results in software development task estimation from textual descriptions?

There are many neural network architectures, and each of them has many hyperparameters. The goal of the neural network is to have the right capacity to catch the relationship between inputs and labels from training data. This research question is about finding the configurations and architectures which lead to the most accurate estimates. Finding the best model might not be possible, but at least highlighting what worked best in this project may be useful for future researchers in this field.

Research Question 5 Can the relationship between task textual descriptions and software development effort be transferred across projects and organizations?

Researchers in the software estimation field have not reached a consensus concerning transfer learning in a multi-organization context. It is not clear if company-specific estimation models are superior to models which are trained on data from several organizations.

Research Question 6 Are the results obtained when training the model on publicly available data applicable to commercial contexts?

Publicly available data in software estimation context are often gathered from open-source projects which are different in how often and how accurately time spent on programming tasks is reported. The labeling coverage in commercial context can be 20 times higher than in open-source context. Results obtained from publicly available data could also be worse because of the amount of noise in the data. It is not obvious if the same models will provide better, worse or the same results in commercial context than in open-source context.

Research Question 7 Is the estimation accuracy provided by the model sufficient to be used by businesses?

Even though the research goal might be achieved, it doesn't entail that the model can be used in commercial contexts if human expert estimation accuracy is substantially better.

1.3 Research Method

To achieve the research goal, three project phases were conducted as shown in Image 1.1. First, both publicly available and classified commercial data was gathered. Then an incremental design and experiment process was conducted on publicly available data, taking into account the results of the author's previous literature studies. In each iteration a new model was designed based on the feedback from the previous iteration. Finally, results from the experiment were validated on two commercial datasets.

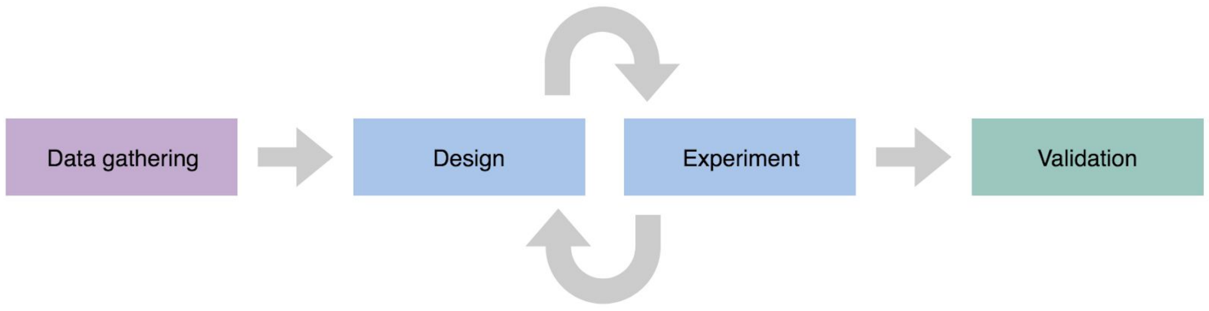


Image 1.1 Research method

1.4 Contributions

In this research work the author has attempted to build a model which can learn the relationship between programming task textual descriptions and time spent on their execution, which is something the author has not found other researchers have done before. Observations indicating transfer learning effects and optimal model configurations and architectures are present in this paper. Not only are the findings and the methods used in this project valuable for further research, but there are also some spillovers. The discovery of publicly available data and the automated methods used in this process is a contribution by itself, and the communication and legal details of obtaining access to commercial data can also be found useful. They can be used in research projects focusing on different properties of programming tasks than logged time.

1.5 Thesis Structure

Chapter 2 introduces the theoretical concepts of word embeddings and recurrent neural networks as well as the current practices and research status in the software estimation field based on previous literature review. The motivation for this project is also further elaborated in Chapter 2. Chapter 3 covers the first phase of the project - data gathering and preparation for the design/experiment phase. Chapter 4 describes the final design of the experiment, and Chapter 5 discusses the final results of the developed model in the context of the Goal and research questions. Chapter 5 also includes validation results in commercial context and threats to validity. Finally, Chapter 6 describes the conclusions and future work.

Chapter 2

Background Theory and Motivation

The author did a structured literature study of more than 200 publications about software effort estimation and reviewed top 10 most popular project planning platforms in software engineering in author's previous work (Tubelis, 2017). This chapter starts by highlighting the most relevant details from the previous research justifying the research direction the author chose to pursue. Further, the reader is introduced with the most important technologies used to achieve the research goal. At the end of the chapter, the most relevant similar research papers ar commercial solutions are briefly summarized.

2.1 Current Estimation Practices in Software Engineering

Software estimation is an important and at the same time very challenging task in software engineering. Predicting the time it will take to implement something that has never been done before is hard. Most often software development effort estimation is done by developers and managers while other methods such as parametric or data-intensive machine learning estimation is less popular. Although humans experience psychological biases e.g. anchoring effects, the industry doesn't really have a good alternative (Jørgenson, 2014).

The uncertainty of estimates is generally accepted, still it is beneficial to improve the accuracy of the predictions. Estimates have great impact on programmer productivity and business efficiency. Underestimated tasks cause delays, cost overruns and project failures while overestimated tasks are no better according to Parkinson's law which says that "work expands so as to fill the time available for its completion" (Galorath and Evans, 2006, p. 34; Parkinson, 1955). The importance of estimates in software engineering has made this problem very popular among researchers. Many have tried different approaches to solve the problem and most of them have been focusing on project-wide prior-start estimation. At the same time the industry practices have changed and many developer teams have adopted agile principles which put more focus on lower level and late estimates of individual programming tasks. Unfortunately, very little research is done on software task estimation and this project is to improve the situation.

Many companies use issue tracking systems such as Jira, which is currently the most popular one (Tubelis, 2017). These systems are used to register tasks and estimates and in many companies also to track time. For each task, a summary and optionally a description is added. When later estimating new tasks, developers and managers can look back and test how much time similar tasks took in the past. Here the author sees the possibility of a machine learning model which could learn from past time spent on tasks and estimate effort of future tasks as shown in Image 2.1. The most popular issue tracking systems in software engineering are open platforms, which means that such machine learning estimator could be easily adopted by installing a plugin. The possibility of easily puting the solution to test provided additional motivation to the author.

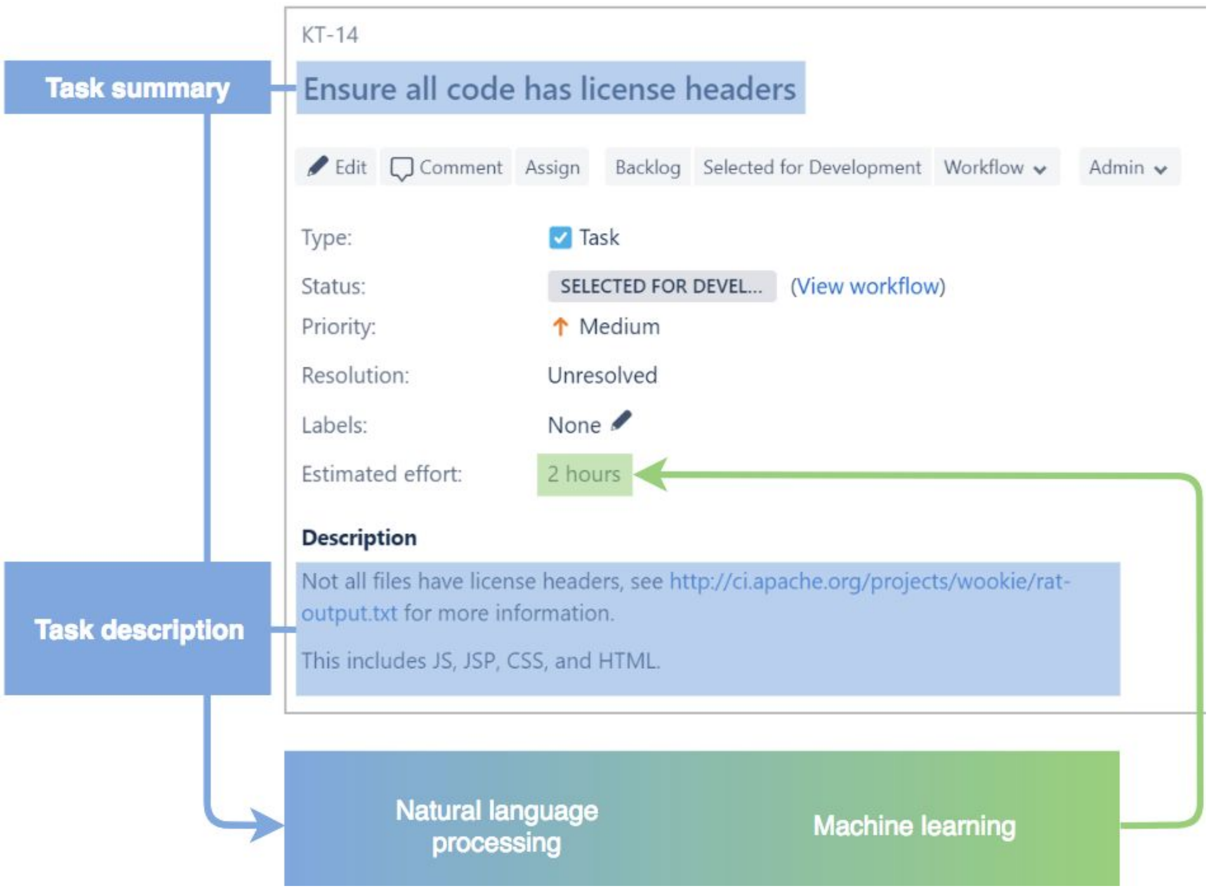


Image 2.1 Concept overview

In agile development, many teams use storypoints instead of hours when estimating backlog items for the next iteration (Usman, Mendes and Börstler; 2015). As described in the last section of this chapter, research has been done trying to learn the relationship between task textual descriptions and storypoints (Choetkiertikul et al., 2018a). However, the author sees a greater potential in using time spent as labels. First, when learning to estimate software tasks from storypoints the estimator will learn the bias of the estimates as storypoints are still estimates and not the actual value. Second, storypoints are different for each team, because a benchmark related to the product the team develops is selected and they cannot be transferred across teams. Third, more data might be accumulated of time spent than storypoints, at least that's what most of the publicly available and commercial datasets used in this research indicated. For more information about the research of software development estimation as well as issue tracking systems, please refer to Tubelis (2017).

2.2 Word Embeddings and Recurrent Neural Networks

To develop such a model as described in the previous section, the author chose a set of established technologies. There is no consensus in the research community about which technology is the most appropriate even for project-wide estimation. Since state-of-the-art neural networks can process text and provide numeric estimates, which is a perfect solution for the problem at hand, author chose to use this method. It also means that developers and managers wouldn't need to add any additional information other than what is already available to make the tool work.

The input to the model would contain task descriptions written in natural language which has to be converted to numeric format in order to pass them to a neural network. The author chose to use **word embeddings** because other similar natural language sentiment analysis problems have been solved applying this method (Maas et al., 2011). If words were represented by one-hot vectors the dimensionality would equal the vocabulary size. However, word embeddings relate terms to each other by their similarity and reduce the dimensionality problem of language modelling (Goodfellow, Bengio and Courville, 2016). This means that similar words appear to close in the vector space. The image below shows that for example among the closest neighbors to "programmer" are the terms "developer" and "hacker" (Tensorflow, 2018). To represent multidimensional vector space in two or three dimensions and create visualizations such as the one below, algorithms such as T-distributed Stochastic Neighbor Embedding can be used (van der Maaten and Hinton, 2008).

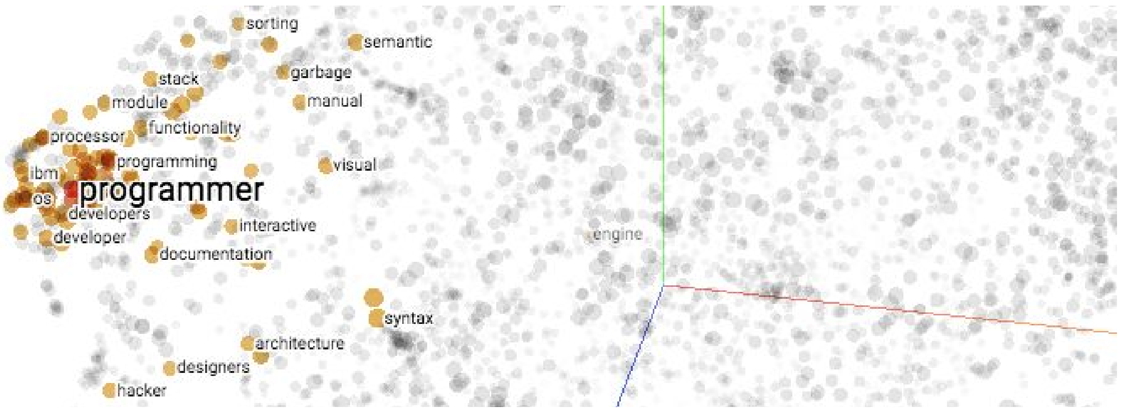


Image 2.2 Three-dimensional visualisation of word embeddings

Word embeddings can be learned at the same time as learning the problem at hand. However, it is often more efficient to pretrain word embeddings on a different tas which exploits the predictive properties of natural language. This is a form of transfer learning and algorithms such as CBOW, skip-gram and GloVe are commonly used as word2vec models. CBOW takes a fragment of text which is defined by the windows length, also referred as context, and tries to predict the word in the middle. Skip-gram does the opposite by predicting context words from a word at hand and weighting the closest words higher (Mikolov et al., 2013). Although CBOW is computationally more efficient, skip-gram is better to learn infrequent words (Google Code Archive, 2013). The task for each algorithm is depicted in Image 2.3.

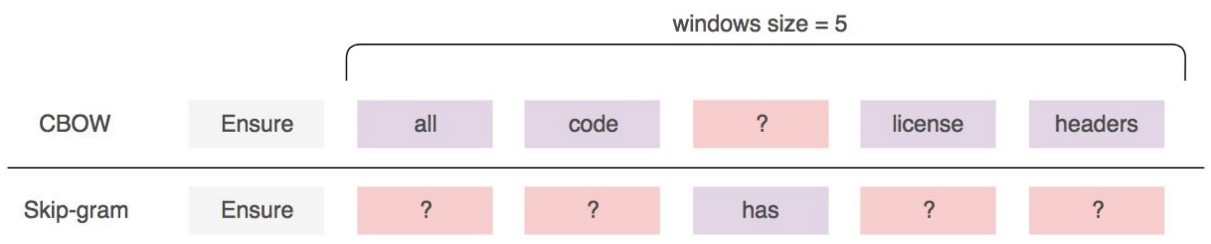


Image 2.3 CBOW and Skip-gram tasks for word embedding learning

Another effective algorithm for embedding learning is GloVe in which such vectors are calculated that their dot vector product equals the logarithm of the probability of their co-occurrence (Pennington, Socher and Manning, 2014).

Once words are converted to their vector representations, they are passed to a neural network to encode the context. Since word sequences are of different length and the order of words matters, **recurrent neural networks** are perfectly fitted for this task (Goodfellow, Bengio and Courville, 2016). Recurrent neural networks feed back some of the hidden state or output of the previous step as an input to the next step. A RNN with a single output is used in this research as shown in Image 2.5. Since LSTM networks have been very successful because of their dynamic time scale of integration enabled by gated input, forget and output which fights the vanishing gradient problem, the author applied the technology to extract context from word embeddings (Hochreiter and Schmidhuber, 1997). A single LSTM cell is depicted in Image 2.4. LSTM networks are composed of many such cells.

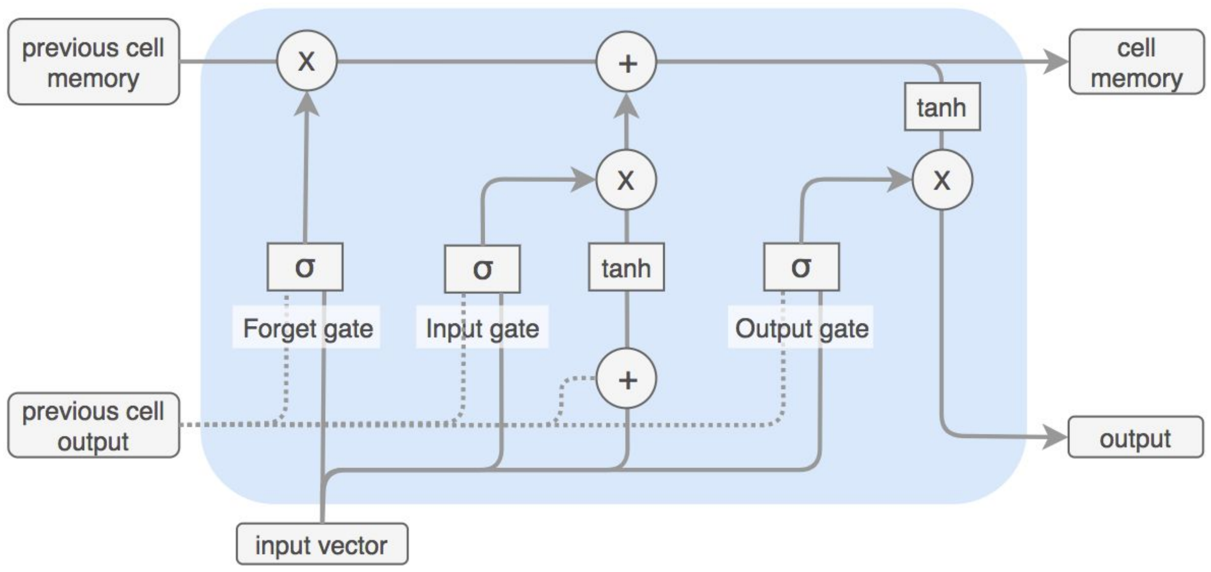


Image 2.4 LSTM cell

In addition bidirectional wrapper was also tried out in order to improve the performance of the LSTM (Schuster and Paliwal, 1997). Since words written in the text can change the meaning of previous words, the sentence is passed to the LSTM twice; first in regular word order and then reversed; then the result is combined together to calculate the final context encoding.

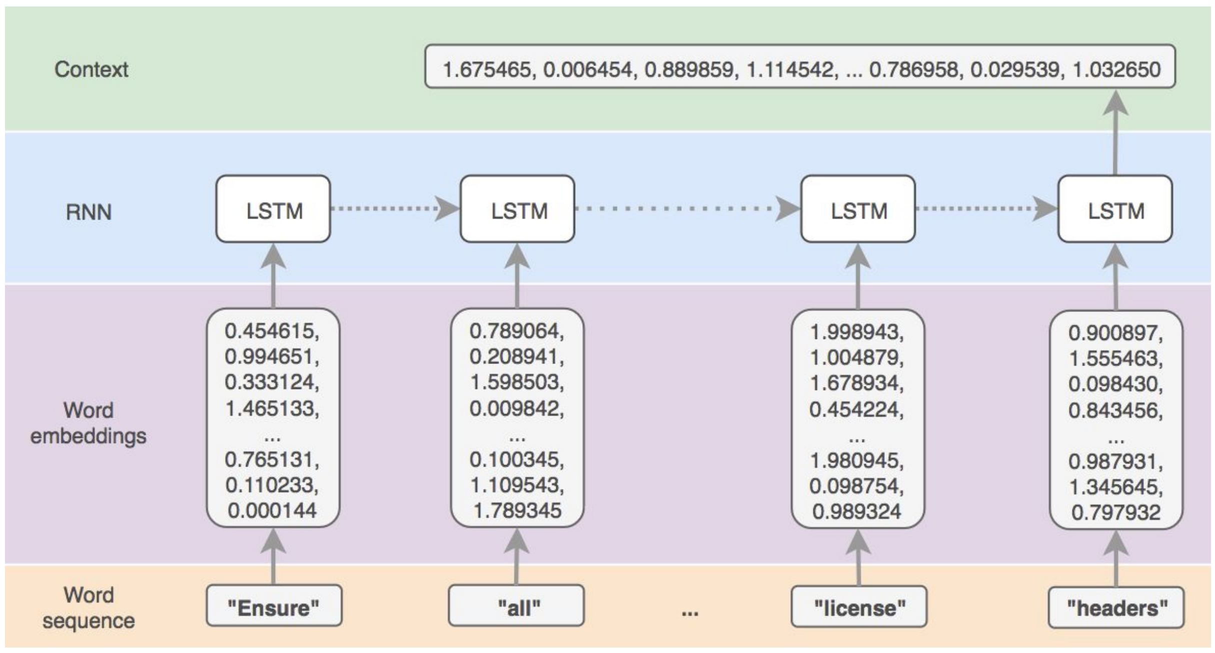


Image 2.5 Unrolled RNN encoding word sequence context

To convert the context embedding to estimates, the author tried out both regular neural networks with dense layers followed by activation functions and single stream highway network as shown in the Image 2.7 because highway networks can be more efficient than plain deep NN (Srivastava, Greff and Schmidhuber, 2015). Highway networks are inspired from LSTM networks and they use gating mechanisms to regulate the information flow e.g. by allowing some information pass the layer unprocessed. Highway networks are composed of many chained highway blocks. A highway block is depicted in Image 2.6.

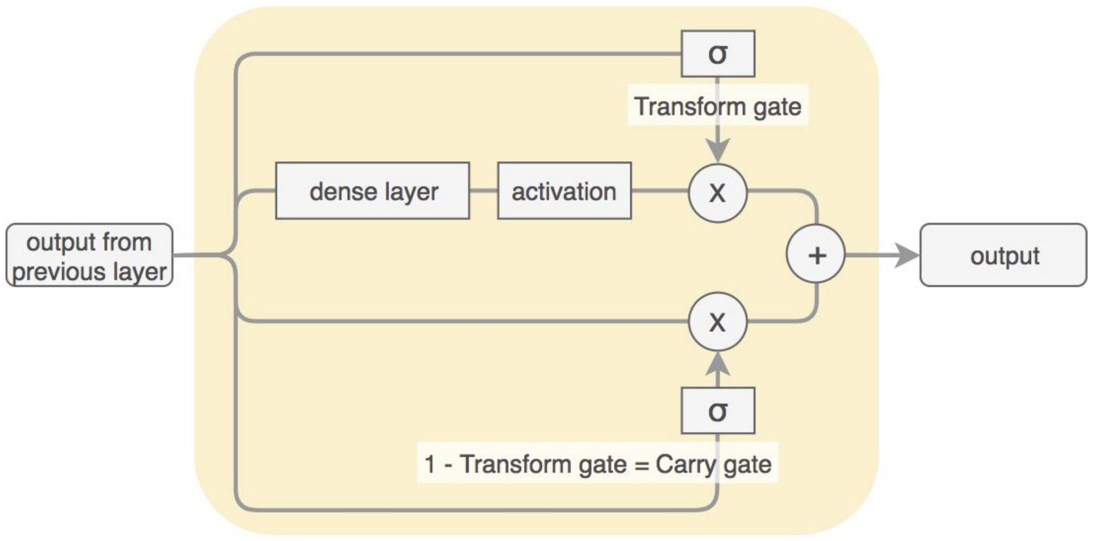


Image 2.6 Highway block

In order to reduce the generalization error of the whole network, dropout regularization were used in most pipeline parts. Dropout regularization is a simple and effective method which prevents overfitting neural networks by turning out random units (Srivastava et al., 2014). A dropout value of 0.25 means that every fourth neuron is randomly turned off.

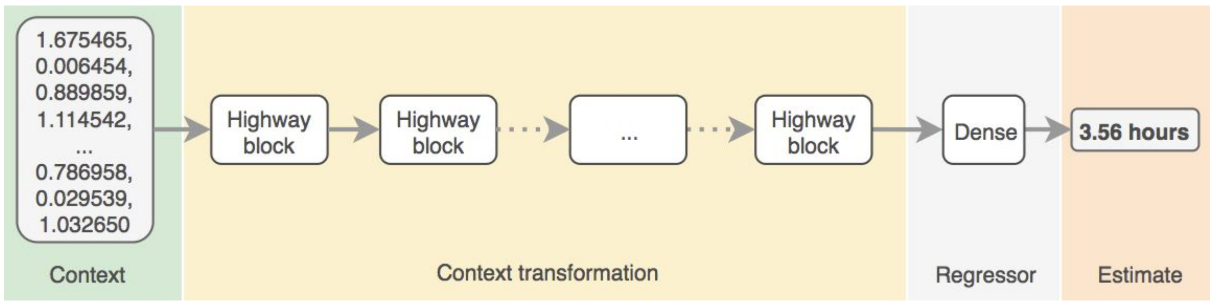


Image 2.7 Transforming context to estimate through highway blocks

In this research RMSProp and its successor Adam gradient descent optimization algorithms were tried out together with the classic SGD optimizer (Hinton, no date; Kingma and Ba, 2015). For hyperparameter optimization TPE estimators method was used (Bergstra et al., 2011). For more information about these methods please refer to the sources.

2.3 Similar Work

Most of the literature research for this project was done in the author's previous research of the author (Tubelis, 2017), however, since then one relevant article was found which had been filtered out in the previous research because of citation count criteria. It's last version was published after the author's previous work was finished. As already mentioned in one of the previous sections of this chapter, Choetkiertikul et al., (2018a) did a similar research project of predicting storypoints for Jira issues. This work and the source code the author published on Github has been used as an inspiration for this research project (Choetkiertikul et al., 2018b). Although not very popular, two Jira plugins have also been developed for estimating software development effort of programming tasks. One of the plugins were discontinued this year, while the other admitted to the author in an email correspondence that the accuracy of the estimates is not sufficient. For more information about the plugins, industry practices and the software effort estimation research landscape please refer to Tubelis (2017).

Chapter 3

Data Collection and Preprocessing

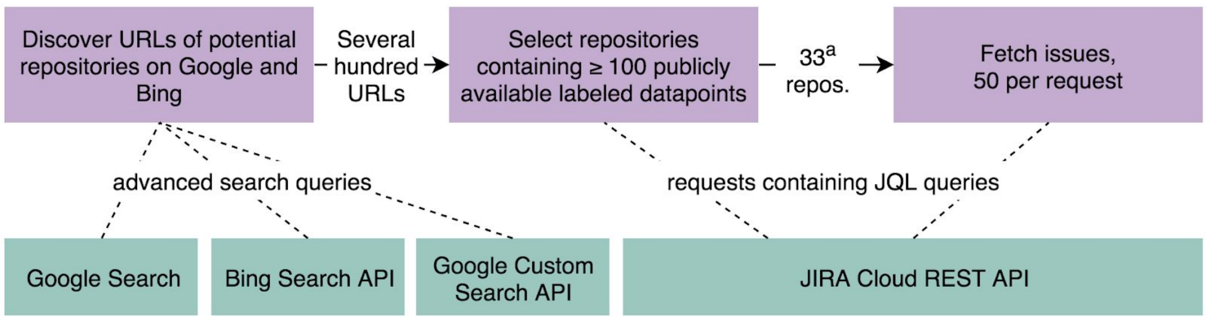
In this research project 32 publicly available JIRA repositories were discovered containing more than 2.000.000 software tasks. Time spent on task execution was reported for more than 63.000 tasks, thus allowing to build a solid dataset for the task at hand. Confidentiality agreements were made with two commercial companies allowing to create a validation dataset of more than 35.000 labeled issues. The author strongly believes that the list of publicly available JIRA repositories is a valuable resource for further research also in other fields than software effort estimation. The methods described in this chapter allow to discover even more JIRA repositories based on various requirements. The specific actions taken to reduce the noise in the textual task descriptions as described at the end of the chapter can be used in projects which do research on JIRA issue fields containing natural language. Communication and legal aspects from the collaboration with the commercial companies can be used as an inspiration for other data intensive research projects.

3.1 Discovering and Fetching Publicly Available Data

Researchers focusing on prior-project effort estimation have been using more than 25 well-known and publicly available datasets (Tubelis, 2017; Wen et al., 2012). However, these could not be benefited from in this project because they were comprised of only parametric project-wide data while the machine learning model created in this project is intended to make estimates from textual descriptions of individual tasks. Therefore the author did an extensive search of publicly available JIRA repositories on the Word Wide Web applying several different methods.

First, the author did an unstructured search on the internet and found a few resources listing public JIRA repositories (Choetkiertikul et al., 2018a; Sloat and Swearengin, 2011). The author observed that the title of most of JIRA repository startpages contains the phrase "System dashboard" and the keyword "JIRA". Through a Google search by keywords "system dashboard" and "public jira" the author found a few more repositories, still looking for a way to do an exhaustive search efficiently, because checking each link manually was very time-consuming. The first optimization was to write a script which can automatically process a list of potential URLs of publicly available JIRA repositories. The implementation of the checker was possible by using Jira Could REST API (Atlassian, 2018). Not only was it possible to test if the particular URL is a link to publicly available JIRA repository, but also how many labeled issues was publicly available in the repository. By labeled issues the author denotes issues which are resolved and the time spent on their execution is greater than zero. In some repositories the user could filter issues by the time spent, but could not actually access this field. Such repositories and those that contained less than 100 issues with accessible labels were discarded.

The author also tried to automate the gathering process of the potential URLs by searching the Web through the APIs of Bing and Google. The Bing Search API worked very well and several new repositories were discovered (Microsoft Azure, 2018). Working with Google Search API was not that easy, because a workaround had to be used, and still only the first 100 search results could be returned even in paid version (Google Developers, 2018; Stack Overflow, 2017). Therefore the author collected all potential URLs manually by searching for websites with title containing "System dashboard JIRA" and then fed them to the checker mentioned earlier to filter out repositories which didn't contain at least 100 accessible labeled datapoints. Both when searching on Bing and Google advanced search queries were used (Microsoft Docs, 2017; Warner, 2015).


Discover URLs of potential repositories on Google and Bing
Several hundred URLs
Select repositories containing ≥ 100 publicly available labeled datapoints
33[a] repos.
Fetch issues, 50 per request
advanced search queries
requests containing JQL queries
Google Search
Bing Search API
Google Custom Search API
JIRA Cloud REST API

[a] only 32 repositories could be used because one had configuration issues discovered in the third step

Image 3.1 An overview of data discovery and fetching process

Once a list of 33 JIRA repositories were collected, the issues had to be fetched so that that they could be fed to the model. The author implemented a script which downloaded the issues by accessing JIRA Cloud REST API (Atlassian, 2018). Labeled and unlabeled issues were downloaded separately by using JQL search query. The API is designed so that only up to 1000 issues can be fetched per request to avoid performance hit (Atlassian Documentation, 2017). For several repositories the limit was lower than that and in others fething 500 or 1000 issues per request caused unreasonably long loading time. After trying out a few different values, the author ended up sending separate request for each 50 records, which worked well for all repositories. The data used in the final iteration of the experiment described in Chapter 4 and Chapter 5 was fetched between May 19, 2018 and May 20, 2018.

In the process of fetching the issues, several exceptions occurred, such as servers denying access or the client losing connection. Such exceptions were handled by awaiting a certain amount of time and trying again. One of the repositories didn't correctly respond to requests containing JQL queries most likely because of a server configuration error (Stack

Overflow, 2014). To save time, the author decided to skip this repository and not implement a workaround because the repository didn't contain that many labeled datapoints as some of the other repositories from the list. Later in the filtering process, which is described in the next chapter, the author found out that issues fetched from one of the repositories and marked as labeled actually didn't contain labels. Since the repository still contained more than 100 publicly available labeled datapoints, the unlabeled labeled issues were moved to unlabeled issues and the repository was not removed from the list.

Finally, to support the last phase of the project, in which the model was validated in commercial context, the author added new functionality to the fetching script allowing to log in the repository to gain access to protected issues. Both API token and the somewhat unsafe approach of typing password in terminal can be used to log in. More information on how access to commercial data was gained is described in the next section of this chapter. Although in most cases the author preferred saving data in JSON format because of its readability, data fetching saved data in CSV format, because one can easily append new datapoints to the file without loading the file in memory. If file is loaded in memory after every request, the script becomes slower by each step.

A summary of the whole data discovering and fetching process is depicted in image 3.1. The script examining potential URLs of publicly available JIRA repositories, the two scripts using Google and Bing Search APIs to gather potential URLs and the data fetching script are made publicly available on GitHub and well documented so that they can be modified and reused in similar future projects (Tubelis, 2018). The actual list of repositories and their features are described in Section 3.4.

3.2 Gaining Access To Commercial Data

Despite the fact that there are many publicly available JIRA repositories, the labeling coverage in those are only around 3%. In addition those are mostly open-source projects where time reporting is not as important as it is in commercial contexts. Developers in commercial organizations have much higher motivation to report the time they spent on programming tasks because their paychecks might be dependent on these values or their management asks them to keep track on how much time they spend on each activity. Therefore it is important to validate the results of this research in a commercial context to prove that they are not impacted by the noise and general quality of publicly available data. As it turned out in the experiment, the labeling coverage in data gathered from commercial companies can be 20 times higher than in publicly available repositories in average. To validate the results got from training the model on publicly available data, the author gained access to confidential data in two companies in Latvia. To find these companies and get their approval the author went through a couple of steps, which are described further in this section and can be used as an inspiration for similar research projects.

The criteria for the companies to be eligible for the project was to have a few thousand labeled issues registered in Jira and the task descriptions had to be written in English. To find such companies, the author did a little survey among his friends who work in different companies in software engineering field. In total 14 people were questioned about 13 different companies they currently work for or have worked for during the last 5 years. The communication was done in a private digital correspondence using LinkedIn and Facebook Messenger and anonymity was granted. As shown in the image below, 2 of them were living in Norway and 12 in Latvia. 10 were currently employed in one of the companies they gave answers about, 3 exited the last company they answered about in 2017 and one before 2017. This information was obtained from the respondent's social media profiles on LinkedIn and Facebook.

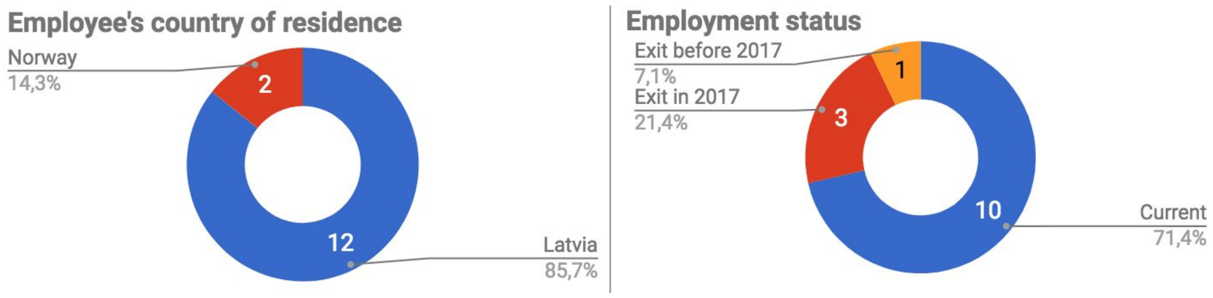


Image 3.2 Respondent country of residence and employment status

Most of the 13 companies the respondents answered about were primarily working in IT industry, and many of them were international as shown in the image below.

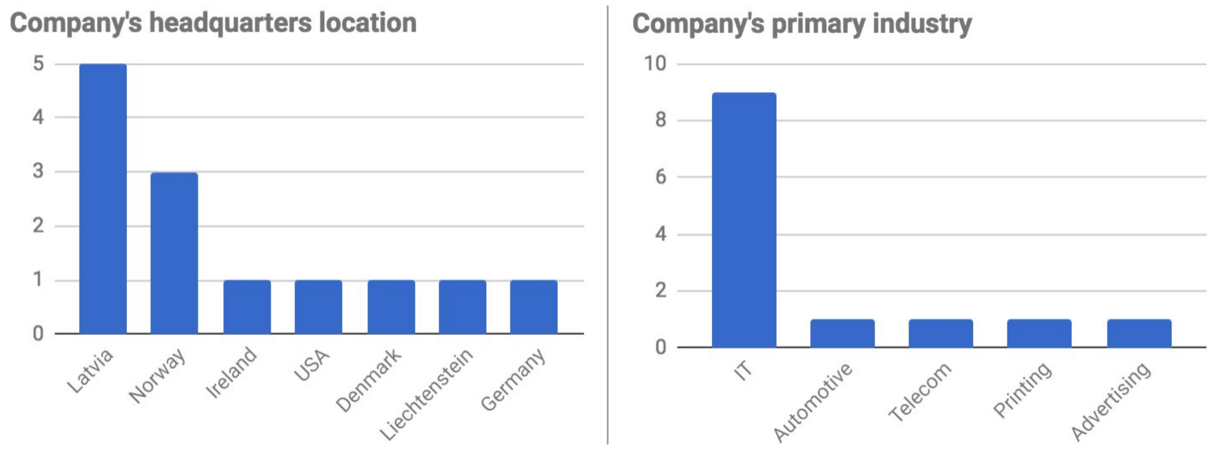


Image 3.3 Company locations and industries

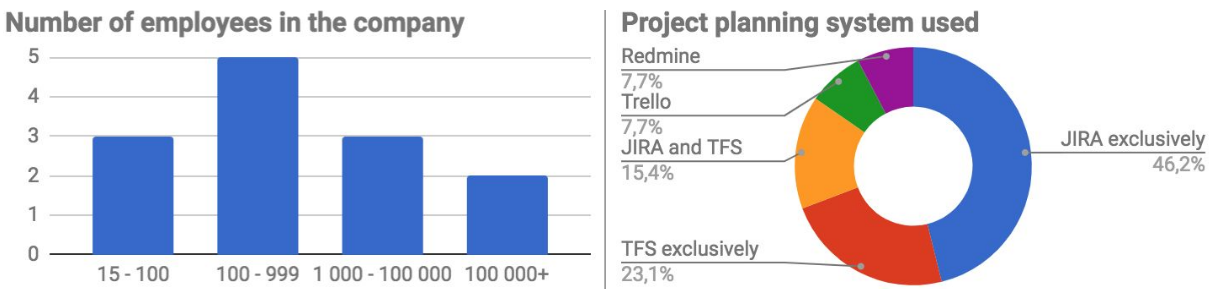


Image 3.4 Company sizes and project planning systems used

The size of the companies was also very different. Although the results of this little survey cannot represent the whole software industry, it gave an impression of the potential of finding

a company to collaborate in this project. At the same time it just confirmed the results of authors previous research showing that project planning platforms are widely used and that JIRA is the most popular one, as seen in the image below (Tubelis, 2017).
As shown in the image below, in some companies time spent on tasks is not reported or is reported in a different system so it cannot easily be linked with the tasks in project planning system. Although English as working language is very common in software companies located in countries where English is not among the official languages, still some companies prefer writing task descriptions in the local language.

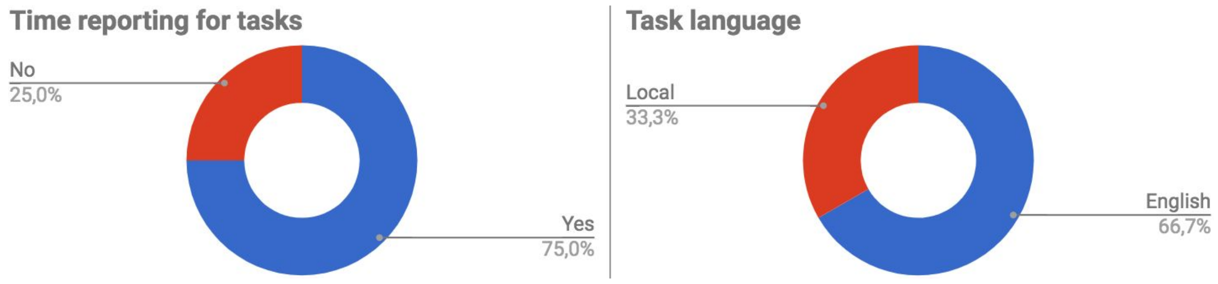
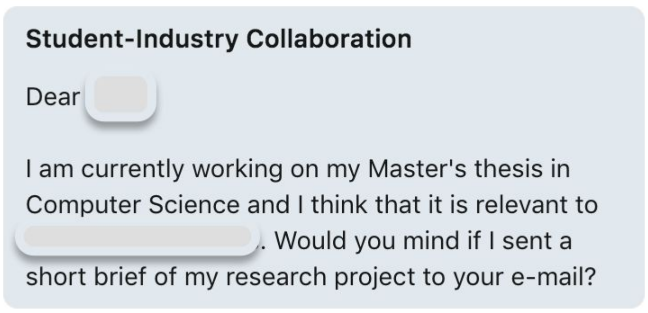


Image 3.5 Time reporting practices and task descriptions language

After the survey, author took contact with the management of the companies which satisfied the requirements. Companies whose employees gave incomplete answers to the survey and other companies were also contacted. The communication was planned in detail to increase the chances of successful result. LinkedIn and Email was used as communication channels and message texts were prepared in beforehand; then tailored for each company to include a personal reference. Before the actual contact, the scenario was tried out with a friend of the author and adjustments were made so that the communication would be as concise and smooth as possible to save the time of the managers and at the same time make the collaboration offer appealing.

The managers were first contacted on LinkedIn by sending a short message like the one in the image below. If the author had a personal reference then the message was started mentioning it. In total 11 managers were approached and 9 managers did reply positively.



Student-Industry Collaboration

Dear [redacted]

I am currently working on my Master's thesis in Computer Science and I think that it is relevant to [redacted]. Would you mind if I sent a short brief of my research project to your e-mail?

Image 3.6 First message approaching a potential collaboration company manager on LinkedIn

Further, a short email message was sent to the managers with a project brief attached. The final version of the project brief which was sent to companies after two of them already had approved the collaboration request is added as Appendix 1 to this paper. In the email correspondence the author found out that several of the approached companies which showed interest to collaborate were not eligible to the criteria. Ultimately, two companies agreed to collaborate and confidentiality agreements were signed.

The brief sent to the companies addressed such important questions as what the project is about, what the company gets if they decide to participate in the project and what are the confidentiality rules. The terms "data" and "accessing data" was purposely avoided so that the managers wouldn't get scared. Jira contains data which the companies are not allowed to publish because they have non-disclosure agreements with their clients. Some of the data can also be used to compromise the competitiveness of the company. Therefore after checking that the company satisfies the eligibility requirements, a strict NDA was proposed.

The importance of a strict NDA was also indicated when one of the collaborating companies approved the request after rejecting it at first just because of much stricter confidentiality rules than proposed initially. Although without the confidentiality rules it would not be possible to test the model in commercial context, they also put serious restrictions on the research activity. The data could not be brought outside the company's office which required some travelling and also didn't give much tries to test the different architectures of the model described in the next chapter because not much processing power was available. The English translation of the NDA is added as Appendix 2.

3.3 Reducing Noise In Textual Task Descriptions

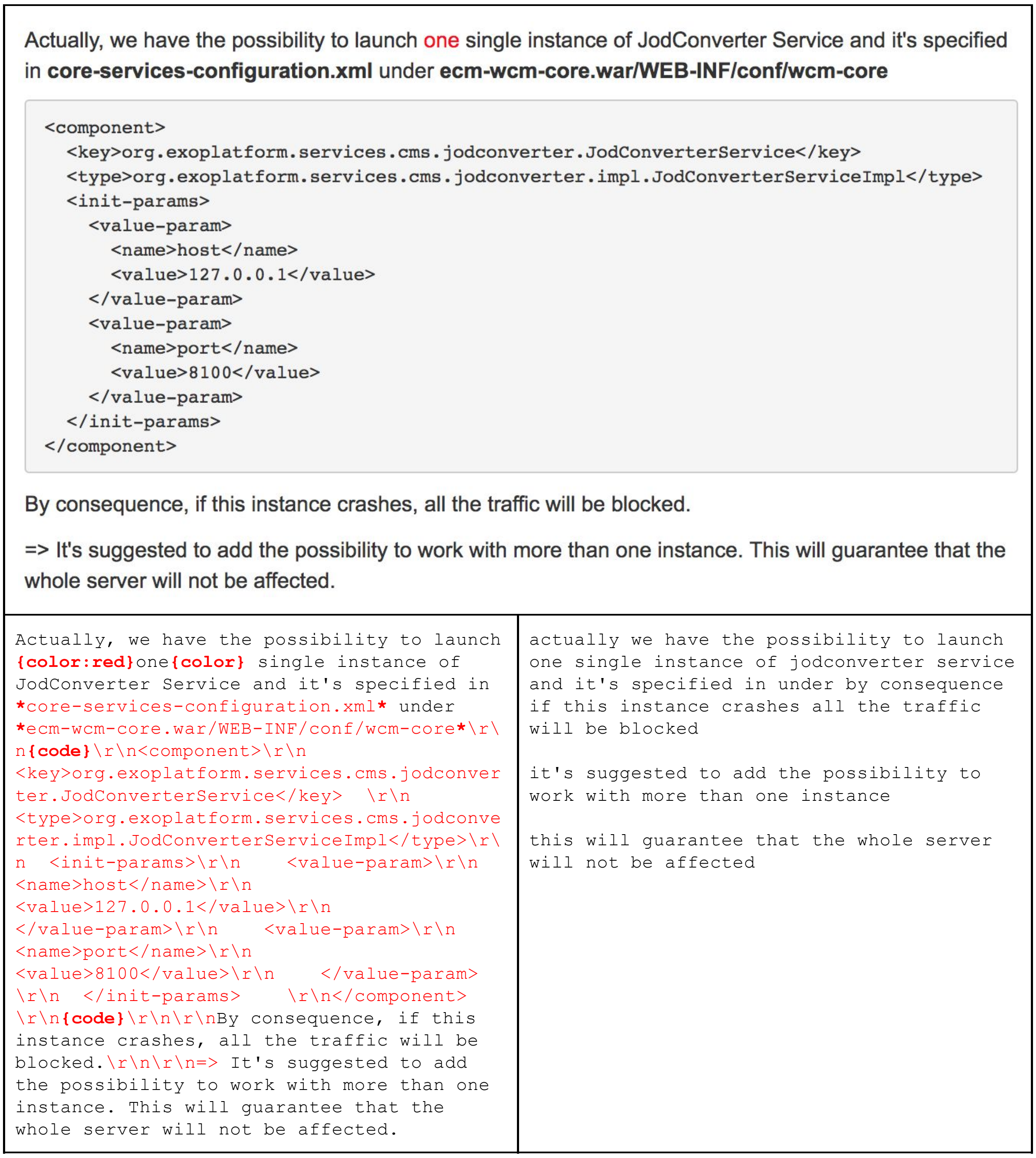
After fetching the data from publicly available JIRA repositories, the author wrote a few scripts to analyze the textual descriptions of tasks and detected that they contained some noise. While the most popular words in the summary field contained stop words in English as expected, the description field contained a lot of numbers and other symbols. After investigating further, the author found out that the textual descriptions contained code fragments, stack trace and formatting tags. The author assumed that this would not be helpful for the model and therefore applied several methods to clean up the text.

First, formatting tags that contain content which is not written in natural language were removed and the text they contained was removed as well. According to Jira markup and observations in issues, tags such as `{code}` and `{noformat}` usually do not contain content in natural language, but rather logs, stack trace and code fragments (JIRA, no date). Second, formatting and emphasis tags and symbols such as colors, bold and italics were removed, but the content was left. Then all tabulators and line breaks were removed as well as internal and external links such as links to other websites and links to Jira profiles marked with special JIRA notation. The author still observed very long descriptions containing stack trace fragments in plain text not marked with any tags. To eliminate these, a regular expression which matched all text fragments containing the keyword "at" followed by one or two words and repeated at least three times. Some issues contained non-English characters, which were removed in the text cleaning process.

All punctuation except trailing period was removed, because trailing period was used at the end of the process to separate the text in sentences. This was done in order to improve word embedding pretraining so that relationships between words in two separate sentences were not considered as the same context. After all of the steps mentioned above were executed, the text still contained a lot of technical identifier, module names and other words that do not belong to natural language. To recognize such words, a special ratio called “alpha ratio” was calculated for each word. The higher the alpha rate the higher the probability that the word does not belong to natural language. The formula used to calculate alpha ratio is:

$$alpha\ density = \frac{count(alphabethic\ characters\ and\ apostrophes)}{count(all\ charaters\ except\ spaces)} \times 100$$

Words with alpha density lower than 93 were removed.



Actually, we have the possibility to launch one single instance of JodConverter Service and it's specified in **core-services-configuration.xml** under **ecm-wcm-core.war/WEB-INF/conf/wcm-core**

```
<component>
  <key>org.exoplatform.services.cms.jodconverter.JodConverterService</key>
  <type>org.exoplatform.services.cms.jodconverter.impl.JodConverterServiceImpl</type>
  <init-params>
    <value-param>
      <name>host</name>
      <value>127.0.0.1</value>
    </value-param>
    <value-param>
      <name>port</name>
      <value>8100</value>
    </value-param>
  </init-params>
</component>
```

By consequence, if this instance crashes, all the traffic will be blocked.

=> It's suggested to add the possibility to work with more than one instance. This will guarantee that the whole server will not be affected.

Actually, we have the possibility to launch {color:red}one{color} single instance of JodConverter Service and it's specified in *core-services-configuration.xml* under *ecm-wcm-core.war/WEB-INF/conf/wcm-core*\r\n{code}\r\n<component>\r\n <key>org.exoplatform.services.cms.jodconverter.JodConverterService</key> \r\n <type>org.exoplatform.services.cms.jodconverter.impl.JodConverterServiceImpl</type>\r\n <init-params>\r\n <value-param>\r\n <name>host</name>\r\n <value>127.0.0.1</value>\r\n </value-param>\r\n <value-param>\r\n <name>port</name>\r\n <value>8100</value>\r\n </value-param> \r\n </init-params> \r\n</component> \r\n{code}\r\n\r\nBy consequence, if this instance crashes, all the traffic will be blocked.\r\n\r\n=> It's suggested to add the possibility to work with more than one instance. This will guarantee that the whole server will not be affected.	actually we have the possibility to launch one single instance of jodconverter service and it's specified in under by consequence if this instance crashes all the traffic will be blocked it's suggested to add the possibility to work with more than one instance this will guarantee that the whole server will not be affected

Image 3.7 Example of text cleaning process

The apostrophe was included in the dividend in order not to remove such words as "it's", which would otherwise have alpha density of 75. Although words do not contain spaces, they were included in the formula because alpha density was calculated for the whole description field and records were sorted by the alpha density ratio in descending order so that it would be easier to detect noisy text.

Finally repeating text fragments up to 15 words were removed and any odd spaces were removed. Odd spaces might be added when replacing several consecutive punctuation marks with spaces. As mentioned earlier, after the text filtering was done, the text was divided in sentences by period marks. An example of the text cleaning procedure is depicted in Image 3.7. Other methods applied to the data in order to remove noise was done when selecting different subsets of the main dataset as described in Chapter 4. The whole filtering process was done by intuition only. Each of the steps described above should be reviewed and tested if they actually improve the estimation accuracy of the model. This was not done because of time constraints and because the models require a lot of processing power and time which is expensive. This is something that should be done in further work.

3.4 Data Insights

After cleaning the textual descriptions from noise, the author analysed the data in order to make better decisions when designing and executing the experiment, draw conclusions and detect coherences between the results and the input data. First the publicly available data and it's most relevant characteristics were compared to the two commercial datasets used for model evaluation. As seen in the table, the labeling coverage is the only major difference between publicly available and commercial programming task descriptions and time reports.

Reference	Text length (words)			Time spent (hours)			Labeled projects	Datapoints		Labeling coverage
	Mean	Med.	Std.	Mean	Med.	Std.		Labeled	Unlabeled	
Publicly available	47	32	58	8.11	3.00	22.43	627	63 474	2 044 801	3.01%
First commercial	55	38	60	7.75	2.50	25.29	530	30 058	25 380	53.89%
Second commercial	42	26	54	4.58	2.00	11.06	140	9 137	5 336	63.13%

Table 3.1 Main characteristics of publicly available and commercial datasets

Further the distribution of time spent, text length and labeled datapoint count in each project was examined. As shown in image 3.8. the distribution of time spent in publicly available Jira repositories reminds more of a log-normal distribution than a normal distribution. This could be a problem for the neural network because there are more examples of tasks with less time spent on them, while larger tasks are more sparse. In addition a log normal distribution means that median estimate will be better than mean estimate if MAE is used as a loss function. The histogram bin size is 5 minutes and it shows that programmers prefer to report even hours, such as 2, 4, 8 and 12 rather than uneven hours and that reporting 10 or 20 minutes past an hour is more popular than reporting 15 minutes past an hour.

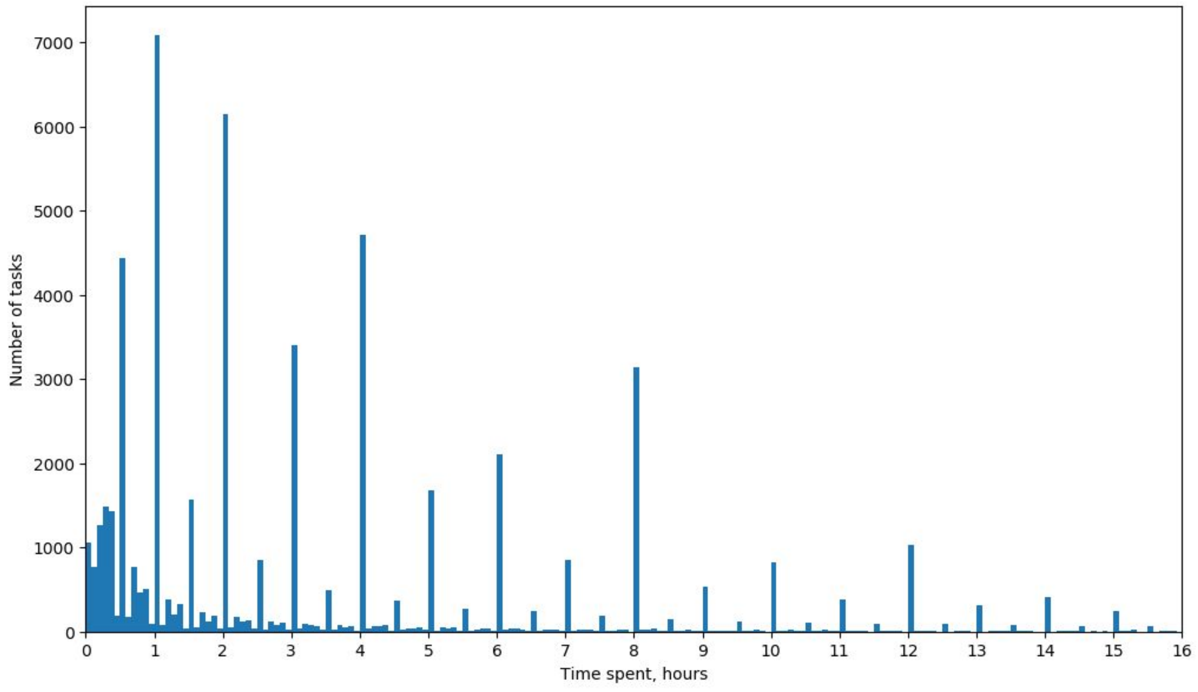


Image 3.8 Time spent on programming tasks up to 16 hours in publicly available data

The most popular time reported is 1 hour which is true in both commercial datasets. The label distribution of the two commercial datasets is shown in Appendix 4. Although some of the other observations mentioned above are not so explicit in the commercial datasets, the are having little or no effect on the model.

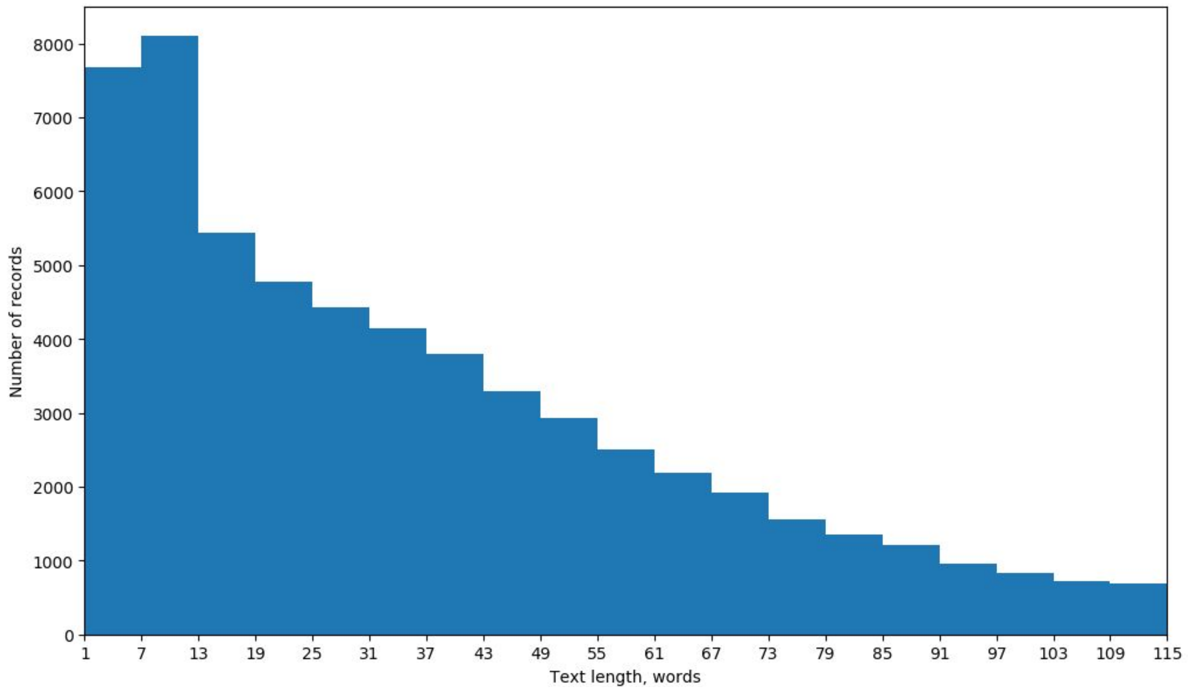


Image 3.9 Task description text lengths up to 115 words in publicly available labeled data

Further, the text length of labeled datapoints was also examined in the main dataset. As shown in Image 3.9, there is a substantial amount of programming tasks with very short descriptions containing only a few words which can make the estimation task very difficult for the model when compared to a human who has much more information such as the code and meetings. It turned out that the first commercial dataset had less proportion of tasks with short descriptions as shown in Appendix 5. However, in the second commercial dataset contained even a larger part of the textual descriptions were up to 7 words long than the publicly available data.

Chapter 4

Incremental Experiment

Numerous iterations of design and experimentation were executed to achieve the research goal. At the start, a simple model prototype was trained on the main dataset. In the following iterations, multiple variations of the model were elaborated. Further, several subsets of the main dataset were created to provide answers to the research questions. Finally, a hyperparameter space was defined and parameter tuning was done, comparing the results to established baselines. Each experiment iteration was followed by a design phase, in which the feedback from the previous phase and new ideas were incorporated in the new version of the model until the final setup, which is described in detail in this chapter, was created.

4.1 Model Architecture and Variations

The architecture of the model was inspired by Choetkiertikul et al. (2018a) and their research. However, the implementation is quite different as the author has used more off-the-shelf libraries instead of operating on a lower level. In addition, several new variations of the model are made to answer to Research Question 3 and Research Question 4. The overall model consists of the following three parts: word embeddings; context network; and context transformation network. A simple regressor is added at the end of the pipeline to obtain the estimate. Each of the three main parts have 2-3 variations as shown in the image below and described further in this section.

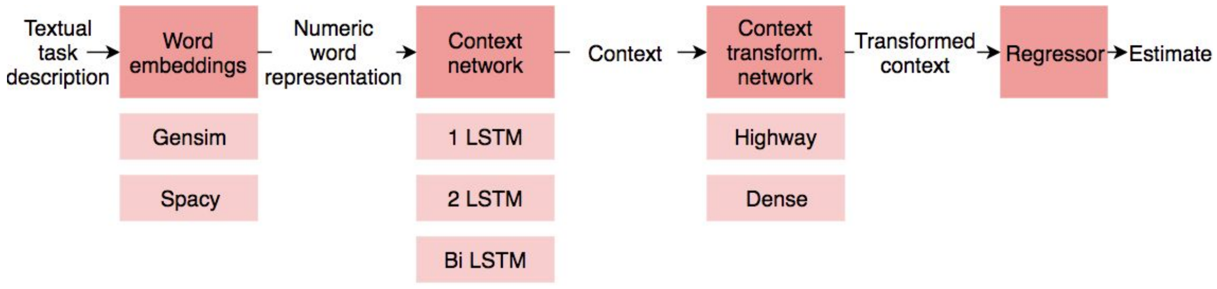


Image 4.1. Model pipeline variations

First, the model converts words to their vector representations, which are learned by performing a different task, such as predicting the previous and the next words in particular text corpus with a certain windows size. In this project, two types of pretrained word embeddings were used. The first alternative was vectors obtained using GloVe algorithm on Common Crawl dataset as provided by Spacy library (Common Crawl, 2018; Pennington, Socher and Manning, 2014; Spacy Models, 2017). These are 300 dimensional vectors trained on texts from World Wide Web. Since these embeddings are trained on general natural language, they may not incorporate the meaning of words specific to programming context or organization. In fact, these vectors covered only about the half of the unique words in the main training dataset, although they include embeddings for 1.1m English words. In addition to that, Choetkiertikul et al. (2018a) used much fewer vector dimensions in their experiment showing that a simpler model may do as well as a complicated one. Therefore the second option is pretrained word embeddings on both labeled and unlabeled issue texts. Since some words may have specific meanings in particular contexts, embeddings were pretrained only on the textual descriptions of tasks from the projects which the model estimated in the particular training session. Both Skip-gram and CBOW methods, which are described in Section 2.2. in more detail, were used as implemented in Gensim library (Rehurek, 2018).

Second, the model used the numeric word embeddings to calculate a context representation using a LSTM network as already mentioned in Chapter 2. Here three versions were used; the first one 1 LSTM was a model where task summary and description were concatenated and then passed to one LSTM network; the second one 2 LSTM was inspired from Choetkiertikul et al. (2018a). In the second version summary and description were passed to separate LSTM networks and the two contexts were then merged by averaging the results. In the third version Bi LSTM a single LSTM networks was put in a bidirectional wrapper, which means that each textual task description was passed to the network twice - once in regular word order and once in reversed word order, and then the two outputs were merged together. Third, the context was passed to a deep multi-layer network which transformed it and then passed to the regressor. Two options were used here; Dense was a simple deep dense multilayer neural network with activation layers as the first option; and Highway was a highway neural network, as Choetkiertikul et al. (2018) used in their paper.

Pipeline part	Choice	Comparison pair
Embeddings	Spacy or Gensim?	*spacy-1-hway* and *gensim-1-hway*
Context network	1 LSTM or 2 LSTM?	*gensim-1-hway* and *gensim-2-hway*
Context network	1 LSTM or Bi LSTM?	*gensim-1-hway* and *gensim-bi-hway*
Context network	2 LSTM or Bi LSTM?	*gensim-2-hway* and *gensim-bi-hway*
Context transformation network	Dense or Highway?	*gensim-1-hway* and *gensim-1-dense*

Table 4.1. Answering architectural questions by comparing different model variations

In total there are 24 possible models which can be built, but since testing all of them would require a lot of resources, the number of the models were cut down to five in such a way that each of the alternatives in the model pipeline can be evaluated. The chosen five model architectures were given short names by which they are referred further in the text: *spacy-1-hway*, *gensim-1-hway*, *gensim-bi-hway*, *gensim-2-hway* and *gensim-1-dense*. The table below gives an overview of how all architectural choices can be made based on the results of these five datasets if for simplicity an assumption is made that the separate parts do not have any synergies.

4.2 Training Datasets With Different Homogeneity Levels

To test the capability of the model to learn estimating programming tasks in different contexts and at the same time test if the accuracy of the model improved when filtering out some datapoints which might be more heterogeneous than others, several subsets of the main dataset were generated. The author proposed the following three assumptions to provide answers to Research Question 2 and Research Question 5 and chose the training datasets in such a way that some indications showing if any of these assumptions are true could be made.

Assumption 1 Estimation accuracy increases when estimating several large projects than when estimating many small projects because data is more homogenous in each project.

To indicate if this assumption is true, three types of datasets were generated by excluding projects with less than 1000, 500 and 200 labeled datapoints from the main dataset as well as one type in which datapoints were not filtered based on project size. To indicate that the assumtion is true, datasets with larger project size shall have higher estimation accuracy given that the number of datapoints is similar and the same model is used.

Assumption 2 Estimation accuracy increases when task descriptions contain more text.

Three types of datasets were chosen to get an indication if this assumption is true. One with datapoints containing at least 20 words, another one with minimum word count of 10 and last type in which datapoints weren't filtered out by task description length. To indicate that the assumption is true, datasets with higher minimum number of words in text

	At least 1 labeled issue in each project	At least 200 labeled issues in each project	At least 500 labeled issues in each project	At least 1000 labeled issues in each project
At least 1 word in each task description	**all_1_1** 54 642 labeled 1 435 732 unlabeled	**all_200_1** 41 686 labeled 374 039 unlabeled	**all_500_1** 32 754 labeled 250 599 unlabeled	**all_1000_1** 23 768 labeled 145 730 unlabeled
At least 10 words in each task description	**all_1_10** 44 499 labeled 1 289 006 unlabeled	**all_200_10** 32 598 labeled 313 850 unlabeled	**all_500_10** 24 585 labeled 179 790 unlabeled	**all_1000_10** 16 945 labeled 118 635 unlabeled
At least 20 words in each task description	**all_1_20** 35 966 labeled 1 106 787 unlabeled	**all_200_20** 25 145 labeled 237 287 unlabeled	**all_500_20** 19 992 labeled 158 405 unlabeled	**all_1000_20** 9 686 labeled 52 840 unlabeled

Table 4.2 Names and datapoint counts in training datasets testing assumptions 1 and 2

descriptions shall have higher estimation accuracy given that the number of datapoints is similar and the same estimation model is used.

As shown in Table 4.2. twelve datasets were created to test Assumption 1 and Assumption 2. The table gives an overview of the different training datasets and the names used to refer to them further in the text is provided in the table below. The chosen datasets cover a wide range of dataset sizes ranging from 54 642 to 9 686 labeled datapoints and therefore can also help answering the second part of Research Question 1 about the number of datapoints necessary to train the model to achieve accuracy which is significantly better than the minimum of mean and median estimate loss baseline described in Section 4.3.

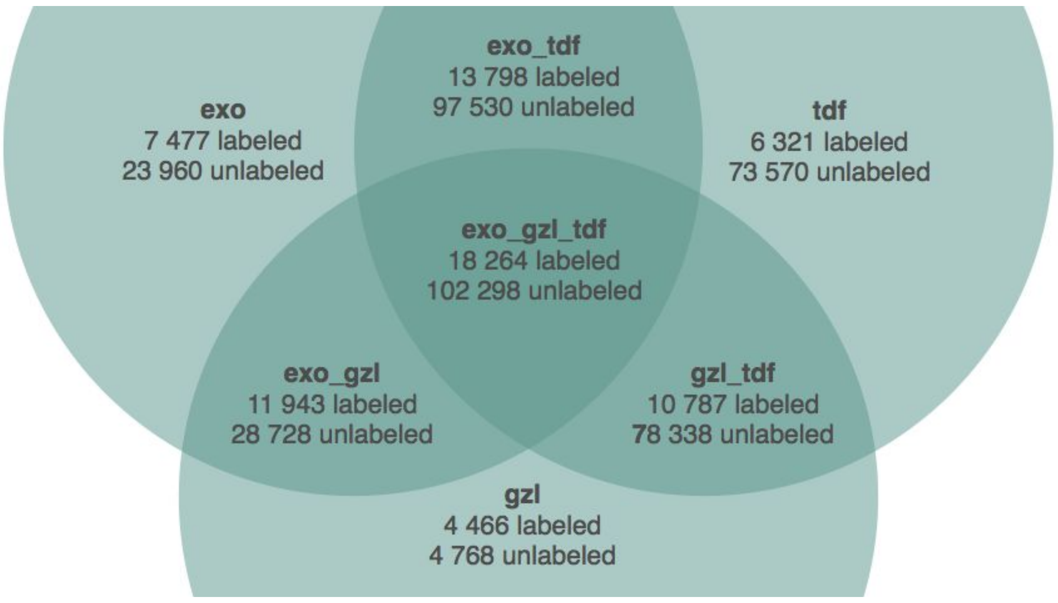


Image 4.2 Names and datapoint counts of training datasets testing Assumption 3

Assumption 3 A model trained on several datasets can provide more accurate estimates than if a separate model is trained for each dataset.

To test this assumption, the three datasets with the highest number of labeled datapoints and all the possible logical relations between them were selected as training datasets. The seven resulting datasets and their sizes are shown in Image 4.2. All issues containing less than 10 words in task textual description were discarded from these training datasets. To indicate that Assumption 3 is true, the same model trained on a merged dataset should have higher accuracy than that of two separate models for each individual datasets, weighted by the number of datapoints in it.

From the experiments conducted in the companies, the author learned that some issues in JIRA do not have clear acceptance criteria and are intentionally kept open for a long time. The time spent on their execution might be logged there for several months or even years, for example to track project management effort. In addition, the author found out that some publicly available JIRA repositories contain test projects where people have logged extreme values, such as twenty years. To increase the homogeneity of the dataset and remove outliers, labeled issues with time spent less than 10 minutes and more than 16 hours were discarded in all training datasets. Furthermore, issue descriptions containing more than 100 dictionary words were truncated if a single LSTM network was used. In the architecture with separate LSTM networks for summary and description, they were truncated by 15 and 95 words because these are the the 90th percentiles of their respective text length rounded up or down by 5 words according to the distribution in the main training dataset.

To test if the model weights were not overfitted to the training dataset examples, 20% of the training dataset was used for model evaluation after each training epoch. In a similar manner, to make sure than the model hyperparameters were not overfitted to the training data, another 20% was used for validation. This means that only 60% of the training dataset was used to learn the model. When training the model, data was shuffled in each of these three groups of data at the time ensuring that the chosen training datapoints were added chronologically earlier than testing datapoints, and none of the validation datapoints were added to their respective JIRA repositories before testing datapoints. In other words, data was sorted by issue identificators in a descending order and then divided in training, testing and validation datapoints, because in real life a company would learn from early programming task actual effort and then estimate later tasks, not the other way around. When ordering and shuffling the data it was ensured that the number of datapoints in training, testing and validation of a particular project was proportional the project size. This helped to avoid situations when all datapoints from a single project end up in training set and none in training and validation set.

4.3 Loss and Baselines

The main loss function used in this project was mean absolute error. This decision was made after observing how the model learns from the training data by comparing mean absolute error with mean square error. When using MSE, the model first outliers first, because of the square factor in the function, however MAE learned all datapoints independently on how far they were from the median or average time spent. Choetkiertikul et al. (2018a) also used MAE as the main loss function which provided additional motivation to use it as the main loss function. Although the literature suggests to use several loss functions when comparing software estimation models because of their individual disadvantages, the author believes that one loss is sufficient to indicate if the methods applied to the software estimation problem in this research can solve it (Tubelis, 2017). Due to time constraints other loss functions and frameworks were not applied in this research project.

To evaluate the results, a main baseline was chosen. Unlike in the research done by Choetkiertikul et al. (2018), random estimate baseline was not used, because it proved to always be worse than the main baseline used in this paper. The main baseline, which the results were compared to, were either mean or median estimate loss, whichever was lower for the particular dataset according to the formula:

$$main\,baseline = min(MAE(mean(train_Y), test_Y), MAE(median(train_Y), test_Y))$$

where $MAE = \frac{\sum_{i=1}^{n} |true\,value_i - prediction_i|}{n}$

To simplify the notation, if only one prediction is passed to the MAE function, it is used for all true values. When evaluating the models, the median and mean values were calculated from the training data, but the actual loss was calculated on testing and validation labels so that the results would dependent less on the difference of training and testing or validation label mean and median values. The goal of the baseline was to indicate if the model would not become agnostic to the input data and draw some correlations. The performance of a model was expressed by the percentage of how much lower the loss of the model was than the baseline loss according to the following formula:

$$model\,performance = 1 - \frac{MAE(model\,predictions)}{main\,baseline}$$

For informative reasons, a human score was calculated. According to Jørgensen (2014) humans estimate software tasks with 30% MAPE. Therefore, the human estimate loss values were calculated accordingly assuming that all estimates are made with 30% deviation. The human score was calculated by the following formula:

$$human\,score = \frac{main\,baseline - MAE(model\,predictions, y)}{main\,baseline - MAE(y \times 130\%, y)}$$

This entails that a model with human estimation accuracy would achieve a score of 100 while a model which is agnostic to the input because it is not capable of extracting the relationship between the input data and expected results and therefore predicts the optimal value would receive a human score of 0.

4.4 Hyperparameter Optimization and Final Validation

In order to make sure that the error is not caused by unfortunate hyperparameters and at the same time test how volatile the model is to hyperparameter variety, Tree of Parzen Estimators algorithm was utilized for parameter tuning as mentioned in Section 2.2. The algorithm requires to define the input parameters as probability distributions or choices and then generates a few random configurations. Later, based on the results of the initial random search, new configurations are generated. This algorithm was preferred because each evaluation round was expensive to calculate and the search space is very large. Models using Spacy embeddings were tuned by running 150 TPE iterations. Since the search space for models using Gensim vector embeddings were larger because it included pretraining parameters, those models were optimized with 200 iterations. For each iteration the model was run until it it didn't show an accuracy improvement higher than 3 minutes in the last 5 epochs. Several parameters such as the batch size of 512 was fixed. The best result according to the error on test datapoints was taken as the final result of the iteration. In a similar manner the best result of the whole evaluation round was chosen by the highest validation performance. In order to indicate the volatility of the model architecture to configuration changes, result confidence were calculated as follows:

$$confidence = min(validation\,results)/average(validation\,results)$$

To confirm that the results from the experiment on publicly available data also applies to commercial context, two of the five proposed architectures were also run on two commercial datasets. Since the confidentiality agreement didn't let the author use external resources while checking the model on the company's premises, the number of evaluation runs were decreased to 10 for each of the two architectures.

4.5 Remarks on The Experiment

The biggest challenge of the experiment was the bugs in the scripts the author implemented to execute the experiment. Since the author worked alone, it was hard to check the author's own code from a quality assurance perspective. But having errors in the code had a very severe effects on the research process. Errors were discovered after running the models for several days and the errors discovered at the end of the project could not be fixed because of time constraints. The errors that were not fixed are mentioned in Chapter 5. A more often peer review would have been very helpful in a project like this. In addition some of the result collection from several servers were done manually and was prone to human failures. This was solved by implementing a script which check the correctness of the gathered results.

Another hindrance was the availability of computational resources. Even though the author used the server available at the university's Telenor AI Lab, it was very busy with other master student models. Since the access to the resources was weakly regulated, the models runned by the author were interrupted when the server was overloaded, so time had to be spent to implement and execute resuming hyperparameter optimization sessions. However, when the author learned that with such a busy server the experiment cannot be finished in the time frame given for masters thesis, the author chose to use Amazon AWS EC2 and Google Cloud virtual machine instances at the author's own expense.

The third problem was the efficiency of the frameworks used in the experiment. The author used Keras and Hyperopt as the main framework for implementing neural network models and tuning the hyperparameters. The first problem was leaking memory between each evaluation which was solved by creating a new TensorFlow session for each evaluation run. Second, neither Keras nor Hyperopt has a great multiprocessing support which didn't really allow to use all the resources available. Therefore a data generator was implemented, which introduced a serious error, and several models were run on the same server simultaneously to achieve higher resource utilization. Not only did some of the libraries have deficiencies, also running the model on different operating systems, such as OS X, Windows 10, Linux and Chromium OS led to platform-specific problems for which workarounds had to be implemented. Even though some problems arose from using Python libraries for various tasks and running the model on different OS, ultimately they let the author be more productive than if more things had to be implemented from scratch and only one OS could be used.

From the neural network perspective an interesting observation was that the model achieved better results when trained on data in hours instead of labels expressed in seconds or normalized from the range of 0 to 1. This is most likely related to the weight initialization values. An additional challenge was to fix all the random seeds of the models, and although the author tried, full reproducibility was never achieved.

Chapter 5

Results and Evaluation

The computationally expensive experiment conducted as a part of this research project consisted of running 150-200 hyperparameter optimization rounds on 5 different model architectures on 19 subsets of the main training dataset. Then two of the five architectures were each tested on the two commercial datasets provided by the collaborating companies. When evaluating the model in commercial context, 10 hyperparameter optimization rounds were run for each architecture and dataset. This resulted in 18,110 evaluation rounds in total each taking from a few seconds for the easiest configurations and smallest training data subsets to several minutes for the most complex configurations and largest data subsets. This chapter describes the results of the research by putting them in the context of the Goal and providing answers to the Research Questions described in Section 1.2. as well as describing the threats to validity.

Goal Construct a neural network model which estimates software development task effort from textual descriptions in English significantly better than mean and median baselines given the actual time spent on task execution.

Solution In order to choose the best setup the author tested 5 different model architectures on subsets of the main dataset composed of data from 32 Jira repositories. Results from testing the models on data from the three largest individual repositories by labeled datapoint count and their logical combinations were not taken into account in questions other than Research Question 5 because the results were too dependent on the choice of testing and validation datasets due to the small size of the repositories. For the model architectures using pretrained word embeddings on Jira task description text corpus 200 evaluation runs were executed for parameter tuning while 150 runs were executed on the model using Spacy embeddings since the *spacy-1-hway* model search space had eight hyperparameters as opposed to 13-15 parameters for *gensim-* models. Model hyperparameters were tuned until the MAE didn’t increase for more than 3 minutes during the last 5 training epochs.

The results depicted in Table 5.1 show that models using word embeddings pretrained on programming task descriptions and employing highway neural network for context transformation performed best, and the best result and the highest confidence was achieved by the model using bidirectional LSTM for context encoding. Therefore this architecture was chosen for statistical testing. The tradeoff of dataset size and homogeneity was explored and the results show that the best improvement as compared to the minimum of mean and median estimate for four of the five model architectures was achieved when filtering out tasks with textual descriptions shorter than 10 words and additionally eliminating projects containing less than 500 labeled datapoints.

Dataset	spacy-1-hway	gensim-1-hway	gensim-bi-hway	gensim-2-hway	gensim-1-dense
all_1_1	6.36%	7.26%	7.50%	7.40%	7.90%
all_1_10	5.95%	7.25%	7.63%	7.70%	6.55%
all_1_20	5.71%	7.35%	7.14%	8.06%	6.43%
all_200_1	5.82%	8.17%	8.31%	7.71%	7.91%
all_200_10	6.60%	9.20%	8.67%	8.59%	8.32%
all_200_20	6.78%	8.81%	8.91%	8.08%	8.09%
all_500_1	5.52%	7.64%	7.66%	8.08%	7.36%
all_500_10	6.69%	9.92%	9.39%	9.49%	9.13%
all_500_20	7.46%	8.55%	8.80%	8.69%	8.33%
all_1000_1	5.49%	7.69%	8.37%	7.00%	7.47%
all_1000_10	5.92%	7.69%	7.95%	6.88%	6.69%
all_1000_20	4.52%	3.65%	4.32%	6.15%	2.20%
Average	6.07%	7.76%	7.89%	7.82%	7.20%
Confidence	97.51%	97.25%	98.19%	98.06%	94.20%

Table 5.1. Model results compared to the best prediction of mean and median estimate baseline

In order to test if the result obtained was significantly better than the baseline, *gensim-bi-hway* model was run once again on *all_500_10* dataset, but now until the result on the training dataset didn't show any improvement for five consecutive epochs. The Wilcoxon signed-rank test p-value for the model estimate deviations on validation datapoints (n=4917) compared to baseline estimate deviations was 4.30e-43 which proves that the model estimates were significantly better than the baseline estimates. Since (a) an error which truncated testing and validation subsets to the floor multiple of batch size was fixed after the hyperparameter optimisation process; and (b) although the author tried but didn't achieve full reproducibility by fixing random seeds; the result when running the model with the same configuration for the second time was 8.27% which was lower than the one achieved in the parameter tuning process. The training (blue) and testing (green) curves are shown in Image

5.1 and the testing and validation predictions are visualised in Image 5.2. The human score of the model was 14 as shown in Image 5.5.

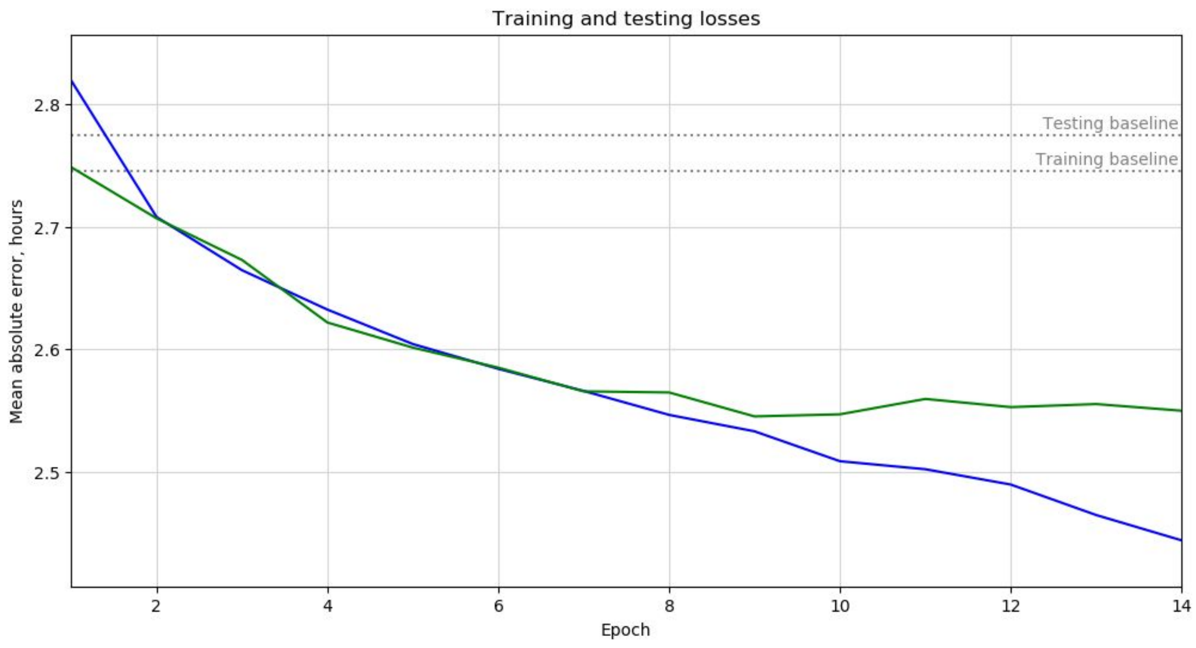


Image 5.1 Training (blue) and testing (green) loss curves of *gensim-bi-hway* model

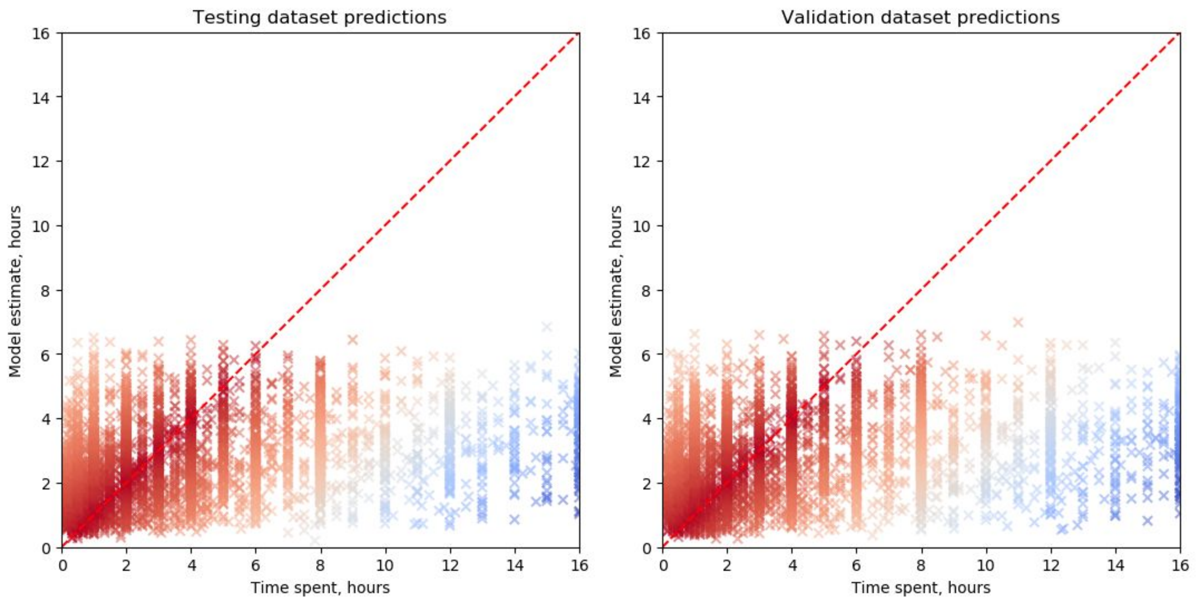


Image 5.2 Testing (left, n=4917) and validation (right, n=4917) prediction plots

Result The research goal is achieved since the the model estimates are significantly better than baseline estimates.

Threats to validity The number of parameter tuning rounds was not proportional to hyperparameter search space sizes. For example *spacy-1-hway* model was run with 150 evaluation rounds on each dataset with search space consisting of 2 uniformly distributed integer parameters (45 possible values each), 4 uniformly distributed floating point parameters (infinite number of possible values) and 2 choice parameters (each with 2 possible values). At the same time the search space for *gensim-2-hway* model was substantially larger as shown in the table below, but the number of evaluation rounds was increased only by 50, which means that a smaller partition of the possible combinations could be tested. Initially one of the hyperparameters (the optimiser type) had three possible values, but later when observing that it was almost never a good choice, it was excluded from the search space making the search space uneven even among the same architecture.

Model	Hyperparameter search space			Estim. search space size	Evaluation rounds	Search space coverage
	Uniformly distr. integers	Uniformly distr. float.	Choice			
spacy-1-hway	2: 45 × 45	4: ∞^4	2: 2 × 2	8: 8.1e3 × ∞^4	150	1.9e-2 / ∞^4
gensim-2-hway	6: 45 × 45 × 495 × 14 ×12 × 15	6: ∞^6	3: 2 × 2 × 2	15: 2.0e10 × ∞^6	200	10.0e-9 / ∞^6

2: 3 × 4 denotes 2 parameters having 3 and 4 possible values consecutively

Table 5.2 Parameter tuning search space coverage of *spacy-1-hway* and *gensim-2-hway* models

The results of the three *gensim-...-hway* model architectures were very similar and therefore it is not obvious that *gensim-2-hway* is the best model architecture. In the smallest datasets such as *all_1000_20* which contained only 5,812 training datapoints and 1,937 testing and 1,937 validation datapoints the result was volatile depending on how the data were split. This effect was especially strong because the testing and validation sets were truncated to 1,536 datapoints because of a data processing error which was discovered at the end of the experiment and due to time and financial constraints the models were not rerun.

Even though the statistical test shows that the failure distributions of the baseline and the model are different the prediction plots indicate that the model did not make estimates higher than 7 hours. The author has tried to solve this issue in several ways, still it tends to appear. When training the model and investigating the prediction plots of each epoch, the following was observed: in the first epoch the model predicted only a few values close to median estimate for all datapoints, then gradually expanded the prediction range. While the range was being expanded, the model also learned some of the testing data if it's capability allowed it. Otherwise the points look more like random values in the plot, although the model was consistently outperforming the median estimate which means that the model had learned a few relationships between the input data and the labels. This can mean that there was a weak correlation between the input data and the labels.

Research Question 1 How much labeled data is publicly available, and how much is necessary to achieve the Goal?

Solution An exhaustive search on the web for publicly available instances of the most popular project planning platform in software engineering was done. In total 32 repositories were found containing more than 63,000 labeled and 2,000,000 unlabeled datapoints. Access to two commercial datasets was also obtained, and the author discovered that the labeling coverage in commercial contexts can be up to 20 times higher than in publicly available data. A middle-sized company had more than 2.5 times the amount of labeled data than any of the organizations allowing public access to their data. This shows that for future projects more focus should be put on using commercial data.

To see how the number of datapoints affects the estimation accuracy please refer to Image 5.3, in which training datapoint count (testing and validation datapoints excluded) is linked to the results of the five model architectures. Both data homogeneity and dataset size affect the estimation accuracy, which has to be taken into account when examining the graph because the datasets on which the models were tested had various homogeneity levels. However, it seems that at least 10,000 training datapoints are necessary to get better results. Smaller datasets are also more volatile to how the data is split in training, testing and validation sets.

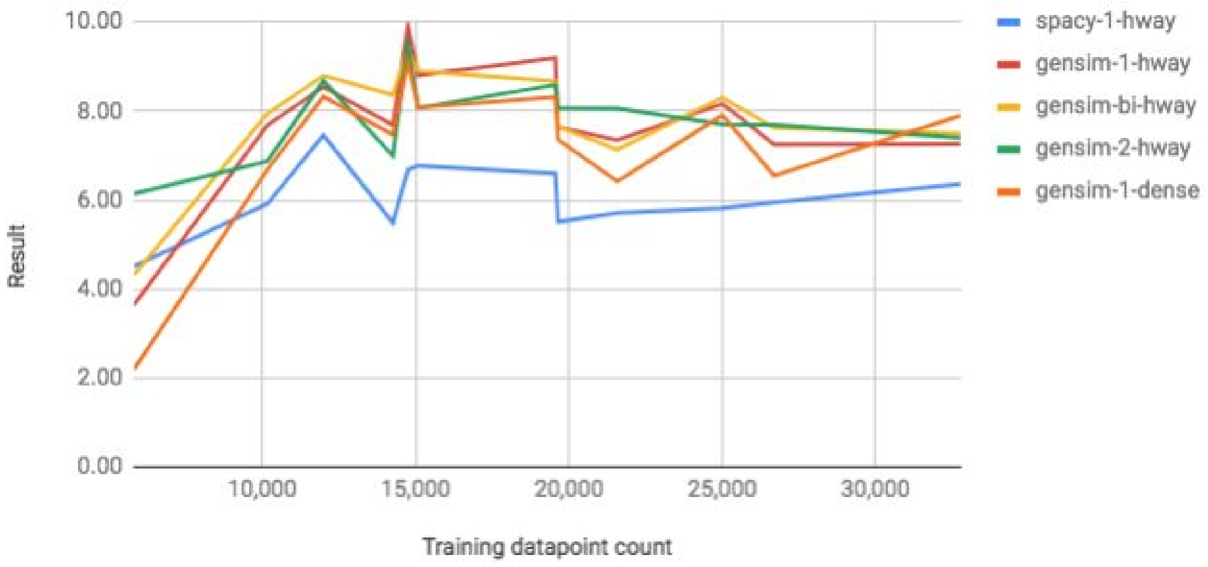


Image 5.3 Training datapoint count and model accuracy

Answer Although around 63,000 labeled datapoints are publicly available and they are not very different from commercial data, even middle-sized commercial companies own a much greater number of labeled data than publicly available repositories because the labeling coverage is much higher in commercial contexts. Using at least 10,000 training datapoints is suggested to achieve credible results.

Threats to validity Although in this project publicly available data was collected from publicly available Jira instances, more data could be found in other publicly available systems which are used to store programming task descriptions, such as Redmine or GitHub. Even when some of the systems do not provide time tracking functionality, there are plugins available that enable this service on these platforms.

<u>Research Question 2</u> **What are the noise and heterogeneity levels in publicly available software development task descriptions and time reports, and how can they be decreased?**

Solution Noise levels in text descriptions can be reduced by examining the text and eliminating odd elements such as stack trace, code fragments and Jira markup tags. Some of the noise in the logged time can be eliminated by excluding outliers because some issues are used to track time of regular activities over a long period of time.

The publicly available data contains data from projects using different technologies, and the time is reported by programmers with various productivity levels. Each organization may have their own style regarding how they report time on issues, such as reporting time only on the main task while executing the subtasks. Such heterogeneity and noise cannot easily be avoided. However, a constraint can be placed on the minimum number of words of task descriptions and the minimum project size. These measures increase the homogeneity of the data and therefore the estimation accuracy is also increased. However, when filtering out data the total number of datapoints is reduced, which negatively affects the estimation accuracy as already mentioned in the answer to Research Question 1. The optimal constraints for the main training dataset used in this research project were at least 10 words long task descriptions and at least 500 labeled issues in a project, as shown in Table 5.3. Datapoint count is written in parentheses after average result of the five models.

	1 words	10 words	20 words	Average
1 MPS	7.28% (54 642)	7.02% (44 499)	6.94% (35 966)	7.08%
200 MPS	7.58% (41 686)	8.28% (32 598)	8.13% (25 145)	8.00%
500 MPS	7.25% (32 754)	8.92% (24 585)	8.37% (19 992)	8.18%
1000 MPS	7.20% (23 768)	7.03% (16 945)	4.17% (9 686)	6.13%
Average	7.33%	7.81%	6.90%	

MPS - minimum project size, the minimum number of labeled datapoints in a project

Table 5.3 Homogeneity and size tradeoff in the main dataset

To show that the results are not impacted by the similarity of the label distributions of testing and validation datapoints, absolute baseline differences were calculated. MAE of *median of testing data prediction* on testing data was chosen as testing baseline and MAE of *median of testing data prediction* on validation data was chosen as validation baseline. The Pearson correlation coefficient between the average result on the dataset and the absolute baseline difference is 0.6, which is a moderately positive correlation. This means that the models performed better on datasets which had higher baseline differences, and the better results were therefore most likely caused by the increased dataset homogeneity and not the label distribution similarities. The median differences are shown in Appendix 6.

Answer Noise in the data can be decreased by removing code fragments, formatting markup etc. from task descriptions and eliminating outliers. Homogeneity can be increased by filtering out tasks with short descriptions and projects containing few labeled datapoints as indicating that Assumption 1 and Assumption 2 might be true.

Threats to validity (A) There is no proof that cleaning the text from code fragments and formatting tags actually increases the prediction accuracy. (B) When proving the effect of filtering out tasks with short descriptions or with small project size the comparing datasets should be of the same size and similar distribution. In the current research *all_500_10* and *all_500_20* datasets cannot really be compared because *all_500_10* contains 23% more datapoints than *all_500_20*. Because of this unfortunate dataset choice one cannot distinguish if the accuracy decrease is because of fewer datapoints or longer descriptions.

Research Question 3 Does learning word embeddings from unlabeled datapoints have an advantage over using publicly available word embeddings pretrained on general English text corpus?

Answer Yes, a model using vector embeddings pretrained on general English texts was outperformed by models using word embeddings trained on text corpus of both labeled and unlabeled datapoints in 12 out of 12 cases.

Threats to validity Only one type or pretrained vectors was used. Others might have higher quality.

Research Question 4 What neural network configurations provide better results in software development task estimation from textual descriptions?

Solution Most of the hyperparameters didn't really converge to one value which would be constantly better than other values on several datasets for any of the architectures. There were not that many bad configurations as the reader might see from the confidence intervals in Table 5.1. The most volatile model to configuration changes was *gensim-1-dense*. The average value followed by the median value after slash and the standard deviation in parentheses of the three best configurations for each model on each dataset and each architecture is shown in the Table 5.4. Each table cell represents 36 configurations (12 datasets × 3 best configurations).

	spacy-1-hway	gensim-1-hway	gensim-bi-hway	gensim-2-hway	gensim-1-dense
Embeddings					
Dimens. (5-500)	300	336 / 349 (102)	290 / 300 (118)	281 / 284 (103)	343 / 374 (114)
Algorithm	GloVe	skip-gram: 36	skip-gram: 31 CBOW: 5	skip-gram: 36	skip-gram: 32 CBOW: 4
Min. repet. (1-15)		9 / 9 (4)	10 / 11 (3)	7 / 7 (4)	9 / 9 (4)
Wind. size (3-15)		11 / 12 (3)	10 / 10 (3)	12 / 12 (2)	9 / 10 (3)
Iterations (5-20)		13 / 14 (4)	14 / 14 (4)	13 / 14 (4)	12 / 13 (3)
LSTM					

Nodes (5-150)	90 / 89 (28)	91 / 85 (35)	80 / 78 (32)	69 / 62 (33)	56 / 53 (33)
Rec. dropout % (0-70)	33 / 35 (19)	39 / 36 (17)	34 / 37 (16)	40 / 38 (17)	29 / 24 (18)
Rec. dropout 2 % (0-70)				25 / 23 (16)	
Dropout (0-70)	34 / 35 (17)	34 / 34 (13)	33 / 29 (18)	51 / 53 (13)	30 / 29 (18)
Dropout 2 (0-70)				31 / 29 (17)	
Merge type			ave: 12 sum: 10 concat: 7 mul: 7	ave	
Context transformation network					
Blocks (5-150)	82 / 79 (37)	87 / 91 (38)	86 / 103 (46)	100 / 109 (39)	13 / 8 (10)
Activation	relu: 16 tanh: 20	relu: 16 tanh: 20	relu: 15 tanh: 21	relu: 22 tanh: 14	relu: 26 tanh: 10
Dropout % (0-70)	34 / 35 (18)	30 / 31 (19)	26 / 18 (21)	32 / 31 (20)	26 / 23 (20)
Optimizer					
Type	adam: 17 rmsprop: 19	adam: 21 rmsprop: 15	adam: 21 rmsprop: 15	adam: 16 rmsprop: 20	adam: 27 rmsprop: 9
Learning rate e-4 (5-50)	32 / 32 (10)	30 / 31 (10)	27 / 26 (13)	29 / 31 (11)	25 / 24 (13)

Table 5.4 Statistics on the model configurations with the highest accuracies

Answer There were no particular configurations that consistently outperformed others. However, it seemed that word embeddings with dimensions from 172 to 457 obtained by skip-gram algorithm worked best. The optimal LSTM node count ranged from 23 to 126, and some recurrent and final dropout seemed to improve the model accuracy.

Threats of validity Hyperparameters should not be averaged as in Table 5.4 because they work in combination e.g. a higher dropout value might compensate for higher node count in LSTM network because one reduces and the other one increases model capacity. Since the results of the best configurations from the three largest repositories were omitted because of the conclusion instability, these configuration preferences might apply only to models trained on cross-company datasets. Another threat is that the predefined hyperparameter ranges might be invalid. In addition, the learning rate parameter might be too greedy to get fast results.

<u>Research Question 5</u> **Can the relationship between task textual descriptions and software development effort be transferred across projects and organizations?**

Solution This result should be considered with caution, please see threats of validity. The results used to answer this question have been excluded from the solutions of the other research questions because of instability. To evaluate the synergies, the three datasets with the largest datapoint count along with all of their logical combinations were selected, and all 5 model architectures were trained to predict task efforts in each of them. Tasks with descriptions shorter than 10 words were excluded. The results are summarized in Table 5.5. They support most of the conclusions made for Research Question 1.

Dataset	spacy-1-hway	gensim-1-hway	gensim-bi-hway	gensim-2-hway	gensim-1-dense
exo-gzl-tdf	5.87%	6.39%	6.89%	6.69%	5.53%
exo-tdf	2.67%	2.68%	4.20%	3.47%	2.79%
exo-gzl	6.45%	6.59%	6.76%	5.55%	4.32%
gzl-tdf	8.45%	10.37%	11.23%	10.26%	9.58%
exo	-1.97%	-0.74%	-0.52%	1.14%	1.19%
tdf	1.57%	2.90%	1.31%	-0.33%	1.80%
gzl	-3.47%	6.76%	5.98%	5.39%	6.95%
Average	2.80%	4.99%	5.12%	4.60%	4.59%
Confidence	97.02%	97.09%	97.10%	97.80%	95.73%

Table 5.5 Results on the three datasets with most labeled datapoints and their combinations

To prove synergy effects when training the same model on several datasets as opposed to having a separate model for each dataset, the weighted accuracy of the individual models has to be lower than the accuracy of the model which is trained on merged datasets. As shown in Table 5.6, in which model accuracies and labeled datapoint counts are depicted, the synergies were present in all cases.

Condition	acc(A)	acc(B)	acc(A) + acc(B)	acc(A&B)	Synergy
exo+gzl < exo&gzl	-0.18% (7 477)	4.32% (4 466)	1.50% (11 943)	5.93% (11 943)	+4.43%
exo+tdf < exo&tdf	-0.18% (7 477)	1.45% (6 321)	0.57% (13 798)	3.16% (13 798)	+2.59%
gzl+tdf < gzl&tdf	4.32% (4 466)	1.45% (6 321)	2.64% (10 787)	9.98% (10 787)	+7.34%
exo&gzl + tdf < exo&gzl&tdf	5.93% (11 943)	1.45% (6 321)	4.38% (18 264)	6.27% (18 264)	+1.89%
exo&tdf + gzl < exo&gzl&tdf	3.16% (13 798)	4.32% (4 466)	3.44% (18 264)	6.27% (18 264)	+2.83%
gzl&tdf + exo < exo&gzl&tdf	9.98% (10 787)	-0.18% (7 477)	5.82% (18 264)	6.27% (18 264)	+0.45%

Table 5.6 Cross-organization transfer learning synergies

The synergy was calculated as follows:

$$synergy = accuracy_{merged} - \frac{\sum_{i=1}^{n} count(datapoints_i) \times accuracy_i}{count(datapoints_{merged})}$$

The total synergy of the three datasets trained with one model as opposed to weighted accuracies of the three individual models each trained on one dataset was +4.79%. This shows one of the possible advantages of this model compared to a model predicting storypoints, because the model can be shared between several organizations.

Answer There are indications of transfer learning synergies when training one model on several datasets approving Assumption 3.

Threats to validity The author observed that because of the limited number of datapoints in the individual datasets, the results were highly dependent on how the training, testing and validation datasets were split. Therefore these results have to be used with caution and retested with larger datasets to obtain conclusion stability.

<u>Research Question 6</u> Are the results obtained when training the model on publicly available data applicable to commercial contexts?

Solution Two of the model architectures were validated on commercial datasets. After filtering out programming tasks where time spent was less than 10 minutes and more than 16 hours, the datapoint count was reduced by approximately 10-20%. Tasks with short descriptions and projects with few labeled datapoints were not filtered out. Ten evaluation rounds were run for each dataset, and the results compared to those of corresponding results on the publicly available dataset are shown in Table 5.7. The *gensim-1-hway* model turned out to have higher accuracy in commercial context, but the accuracy differences with *gensim-bi-hway* model are minimal.

Dataset	Datapoints	gensim-1-hway	gensim-bi-hway	Evaluation rounds
First commercial	26 095	8.40%	8.15%	10
Second commercial	< 10 000	5.00%	4.89%	10
Publicly available (all_1_1)	54 642	7.26%	7.50%	200

Table 5.7 Model evaluation results in commercial context

The model predictions on the testing and validation datapoints of the first commercial dataset are shown in Image 5.4. Here the best model *gensim-1-hway* was used, and the training was done until the testing results stopped improving for five consecutive epochs. Unfortunately, the plot doesn't show strong correlation between the predictions and the actual values.

Answer The model evaluation in commercial context shows that the results obtained in this research are credible, however, they also indicate how dependent the accuracy is on the dataset used.

Threats to validity The number of evaluation rounds are 20 times lower in commercial contexts than when training the model on publicly available data, which means that the models might perform even better in commercial contexts than the data shows. One of the projects in the second commercial dataset contained task descriptions written in Latvian. Due to time constraints, the author didn't exclude that project and it could have impacted the results negatively.

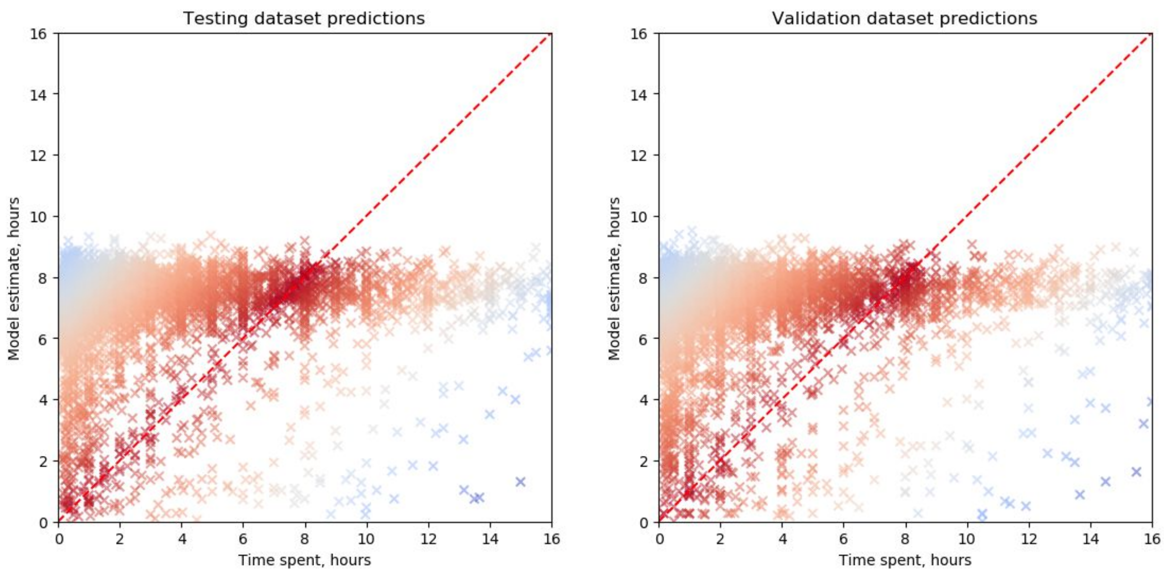


Image 5.4. Model testing and validation predictions of the first commercial dataset (n=5219)

<u>Research Question 7</u> Is the estimation accuracy provided by the model sufficient to be used by businesses?

Answer No, it is not sufficient as the estimates of human experts are much better than the model estimate as shown in Image 5.5.

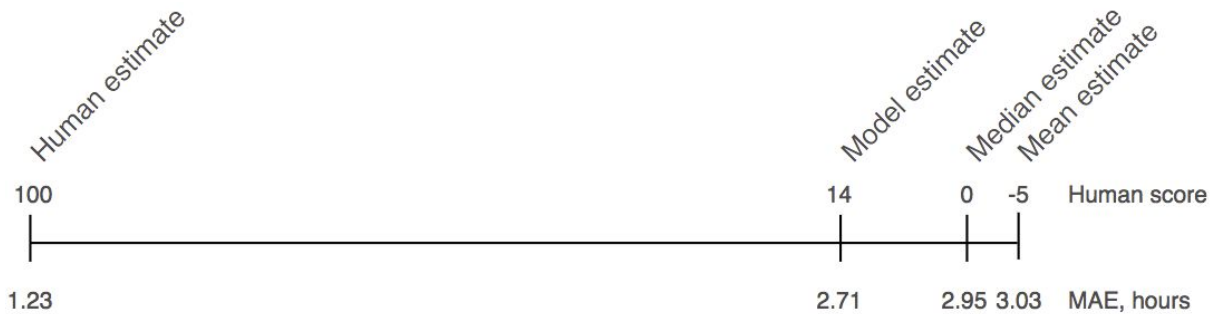


Image 5.5 Model accuracy compared to human estimation accuracy

Threats to validity The author didn't prove that the median estimate is the best single number estimate possible when using MAE. The human estimate accuracy is just an assumption based on research done by Jørgenson (2014), which might differ from organisation to organisation.

Chapter 6

Conclusion and Future Work

The experiment conducted shows that word embeddings and recurrent neural networks can be used to build a model which learns to estimate the time necessary to complete programming tasks given their textual descriptions better than mean and median estimates. However, the estimation accuracy of the model is nowhere close to that of a human expert. There is enough publicly available data available which can be used as a base for further research. The properties of publicly available data are not very different from the properties of commercial data, except for labeling coverage which is much lower. Commercial companies can provide access to great amounts of data under strict confidentiality agreements. The task can be made easier by eliminating tasks with very short textual descriptions and projects with very few labeled datapoints, however this is at the cost of reducing the size of the dataset. The accuracy of the model can be improved by learning word embeddings from unlabeled data as opposed to using pretrained word embeddings on general English text corpus. There are indications that when training the model on data from several companies transfer learning synergies are present.

The future work can be divided into three directions. First, the accuracy of the current model can be increased through incremental improvements. Second, the idea of automated software task estimation can be foregone, and research effort can be spent on improving human estimation practices. Third, ideas from the current model can be used to develop a new model using different technology and architecture.

If the current model is to be improved, here are a few ideas how it could be done:

1. Making the word embeddings dynamic so that their weights get changed when training the model on the estimation task.
2. Using word embeddings trained on general English texts to initialize word embedding pretraining; then improving them by pretraining on the text corpus of all programming task text descriptions.
3. Testing if using all programming tasks to pretrain word embeddings, not only the ones from the projects that are being estimated, can increase the accuracy of the model.
4. Trying GloVe algorithm for embedding pretraining along skip-gram and CBOW algorithms to see if the model accuracy is increasing.

5. First optimizing hyperparameters of embedding pretraining model to minimize embedding loss, and then for the rest of the model in a separate session.
6. Experimenting with different batch sizes and tuning other parameters which are fixed in the current implementation.
7. Focusing more on training the model on commercial datasets after getting some improvements on publicly available data. Gathering more commercial data and doing cross-company training. More publicly available data also could be found on other platforms which allow issue tracking functionality and has time tracking plugins.
8. Although the author prototyped training the model on evenly distributed data, since the datasets were too small it didn't improve the results. However, this idea could be revisited when more data is gathered from commercial contexts.
9. Improving the conclusion stability of the model when training on smaller datasets by using different validation approaches.
10. Narrowing the search space of hyperparameters by 10% margin of the minimum and maximum values of the best configurations. Trying to explore if any relationship exists between the training parameters. Choosing a random fixed value for hyperparameters that don't seem to converge. They may have no impact on the result, but leaving them in the scope increases its size exponentially.
11. Trying to decode the inner state of the neural network to investigate if any improvements can be made based on the diagnostics.
12. Making sure that testing and validation datasets have similar distributions with the training dataset.
13. Eliminating projects with noisy input e.g. project in which task descriptions are written in a different language than in the rest of the repository.
14. Training the model on task descriptions written in other languages. This may provide access to more data.
15. Running the model with several random seeds to test it's volatility to how the data is splitted.
16. Fewer evaluation trials might be used so that the experiment is less computationally expensive, but then need to do research on the number of optimal trials.
17. Using other loss functions such as MAPE or MSE.
18. Investigating how the text cleaning process affects the estimation accuracy of the model. Justifying the choices with accuracy improvements.
19. Make sure this method works in easier task estimating contexts such as when estimating support task severity.
20. Using more advanced architectures employing convolutional neural networks to encode and transform the context.
21. Improving the data fetching script by enabling multiprocessing support and optimizing the memory and processor/GPU usage by profiling.
22. Eliminating cases when description field content duplicates the content of summary field.
23. Improving the reproducibility of the model.

Although the author hasn't spent much time gathering ideas for the second direction of future research, which is improving human expert estimates, combining the first and third direction seems the most promising. Here are some ideas for new approaches on solving the problem in new ways:

1. Providing the model more of the information which is available to human experts. Developing support models which can learn product specifics, productivity factors, technology characteristics and the code base of the product.
2. Using more of the data which is available in Jira, such as the comments and categorical properties of tasks.
3. If the model provides somewhat usable results, invite companies to use the product for free in exchange to their commercial data which can be incorporated in the model, but not exposed to other companies.

Bibliography

Atlassian (2018) Jira Cloud REST API. https://developer.atlassian.com/cloud/jira/platform/rest/ (accessed: 3 June 2018).

Atlassian Documentation (2017) Changing maxResults parameter for Jira Cloud REST API. Available at: https://confluence.atlassian.com/jirakb/changing-maxresults-parameter-for-jira-cloud-rest-api-7791 60 706.html (accessed: 10 June 2018).

Bergstra et al. (2011) Algorithms for hyper-parameter optimization, *NIPS'11 Proceedings of the 24th International Conference on Neural Information Processing Systems,* pp. 2546-2554. Available at: https://dl.acm.org/citation.cfm?id=2986743 (accessed: 15 June 2018).

Choetkiertikul et al. (2018a) A deep learning model for estimating story points, *IEEE Transactions on Software Engineering*, PP(99). Available at: https://doi.org/10.1109/TSE.2018.2792473 (accessed: 30 May 2018).

Choetkiertikul et al. (2018b) A deep learning model for estimating story points. Available at: https://github.com/SEAnalytics/datasets/tree/master/storypoint/IEEE%20TSE2018 (accessed: 16 June 2018).

Common Crawl (2018). Avialable at: http://commoncrawl.org/ (accessed: 31 May 2018).

Galorath, D.D. and Evans, M.W. (2006) Software Sizing, Estimation and Risk Management, 1st edn. Florida: Auerbach Publications.

Goodfellow, I., Bengio, Y. and Courville, A. (2016) *Deep Learning*. Cambridge, MA: MIT Press.

Google Code Archive (2013) word2vec. Available at: https://code.google.com/archive/p/word2vec/ (accessed: 14 June 2018).

Google Developers (2018) Using REST to Invoke the API, Custom Search. Available at: https://developers.google.com/custom-search/json-api/v1/using_rest#search_request_metadata (accessed: 10 June 2018).

Hinton, J. (no date) Neural Networks for Machine Learning, Lecture 6a: Overview of mini-batch gradient descent. Available at: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (accessed: 15 June 2018).

Hochreiter, S. and Schmidhuber, J. (1997) Long Short-Term Memory, *Neural Computation* 9(8), pp.1735-1780. Available at: https://doi.org/10.1162/neco.1997.9.8.1735 (accessed: 14 June 2018).

Jira (no date) Text Formatting Help. Available at: https://jira.atlassian.com/secure/WikiRendererHelpAction.jspa?section=all (accessed: 11 June 2018).

Jørgensen, M. (2014) What We Do and Don't Know about Software Development Effort Estimation, *IEEE Software*, 31(2), pp. 37-40. Available at: https://www.simula.no/publications/what-we-do-and-dont-know-about-software-development-effort-estimation (accessed: 2 June 2018).

Kingma, D.P. and Ba, J.L. (2015) Adam: A Method for Stochastic Optimization, *3rd International Conference for Learning Representations*, San Diego. Available at: https://arxiv.org/abs/1412.6980 (accessed: 15 June 2018).

Maas et al. (2011) Learning word vectors for sentiment analysis, *HLT '11 Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies* (1), pp. 142-150. Available at: https://dl.acm.org/citation.cfm?id=2002491 (accessed: 13 June 2018).

Microsoft Azure (2018) Bing Web Search. Available at: https://azure.microsoft.com/en-us/services/cognitive-services/bing-web-search-api/ (accessed: 10 June 2018).

Microsoft Docs (2017) Bing Web Search API v7 Reference. Available at: https://docs.microsoft.com/en-gb/rest/api/cognitiveservices/bing-web-api-v7-reference (accessed: 10 June 2018).

Mikolov et al. (2013) Efficient Estimation of Word Representations in Vector Space. Available at: https://arxiv.org/abs/1310.4546 (accessed: 14 June 2018).

Parkinson, C.N. (1955) Parkinson's Law, The Economist, November 19h. Available at: http://www.economist.com/node/14116121 (accessed: 9 June 2018).

Rehurek, R. (2018) Gensim: Corpora and Vector Spaces. Available at: https://radimrehurek.com/gensim/tut1.html#from-strings-to-vectors (accessed: 31 May 2018).

Pennington, J., Socher, R. and Manning, C.D. (2014), GloVe: Global Vectors for Word Representation. Available at: https://nlp.stanford.edu/projects/glove/ (accessed: 31 May 2018).

Schuster M. and Paliwal, K.K. (1997) Bidirectional Recurrent Neural Networks, IEEE Transactions on Signal Processing, 45(11), pp. 2673-2681. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.331.9441 (accessed: 14 June 2018).

Sloat and Swearengin (2011) Examples of Public JIRA Instances, Atlassian Documentation. Available at: https://confluence.atlassian.com/display/JIRAHOST/Examples+of+Public+JIRA+Instances (accessed: 2 June 2018).

Spacy Models (2017) Release en_vectors_web_lg-2.0.0 Available at: https://github.com/explosion/spacy-models/releases/tag/en_vectors_web_lg-2.0.0 (accessed: 31 May 2018).

Srivastava et al. (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting, *Journal of Machine Learning Research* 15, pp. 1929-1958. Available at: http://jmlr.org/papers/volume15/srivastava14a.old/srivastava14a.pdf (accessed: 14 June 2018).

Stack Overflow (2014) How to solve “The character '%' is a reserved JQL character” error. Available at: https://stackoverflow.com/a/21985547/9812865 (accessed: 10 June 2018).

Stack Overflow (2017) Programmatically searching Google in Python using custom search. Available at: https://stackoverflow.com/a/37084643/9812865 (accessed: 10 June 2018).

Tensorflow (2018) Embeddings projector - visualisation of high-dimensionality data. Available at: https://projector.tensorflow.org/ (accessed: 16 June 2018).

Tubelis, M. (2017) Examining the feasibility of developing an automated agile task estimation plugin for popular project planning platforms, Specialization Project, Norwegian University of Science and Technology.

Tubelis, M (2018) Bestimate: Estimating JIRA issues using neural networks, Github. Available at: https://github.com/marisst/Bestimate (accessed: 3 June 2018).

Usman, M., Mendes, E. and Börstler, J. (2015) Effort estimation in agile software development: a survey on the state of the practice, 19th International Conference on Evaluation and Assessment in Software Engineering. Nanjing, China, April 27-29, 2015. Available at: https://doi.org/10.1145/2745802.2745813 (accessed: 10 June 2018).

van der Maaten, L. and Hinton, G. (2008) Visualizing Data using t-SNE, Journal of Machine Learning Research 9, pp. 2579-2605. Available at: http://www.jmlr.org/papers/v9/vandermaaten08a.html (accessed: 13 June 2018).

Warner, R. (2015) Google advanced search: A comprehensive list of Google search operators. Available at: https://bynd.com/news-ideas/google-advanced-search-comprehensive-list-google-search-operators/ (accessed: 10 June 2018).

Wen et al. (2012) Systematic literature review of machine learning based software development effort estimation models, Information and Software Technology, 54(1), pp. 41-59. Available at: https://doi.org/10.1016/j.infsof.2011.09.002 (accessed: 10 June 2018).

Appendices

MSc Thesis Project Brief

Project title	Automated Agile Task Estimation On Project Planning Platforms		
Student	Mariss Tūbelis, Computer Science student at NTNU		
Supervisor	Anders Kofod-Petersen, Deputy Director at Alexandra Institute		
Start date	15.01.2018	**Finish date**	11.06.2018

Objective

The objective of the project is to apply state-of-the-art artificial intelligence technology to agile software development task effort estimation. During the project a plugin for JIRA will be developed employing natural language processing and machine learning techniques. The goal is to develop a tool that permits greater accuracy and more desirable qualities than currently available solutions, such as Deckard, Queckt and DEEP-SE.

Model Training

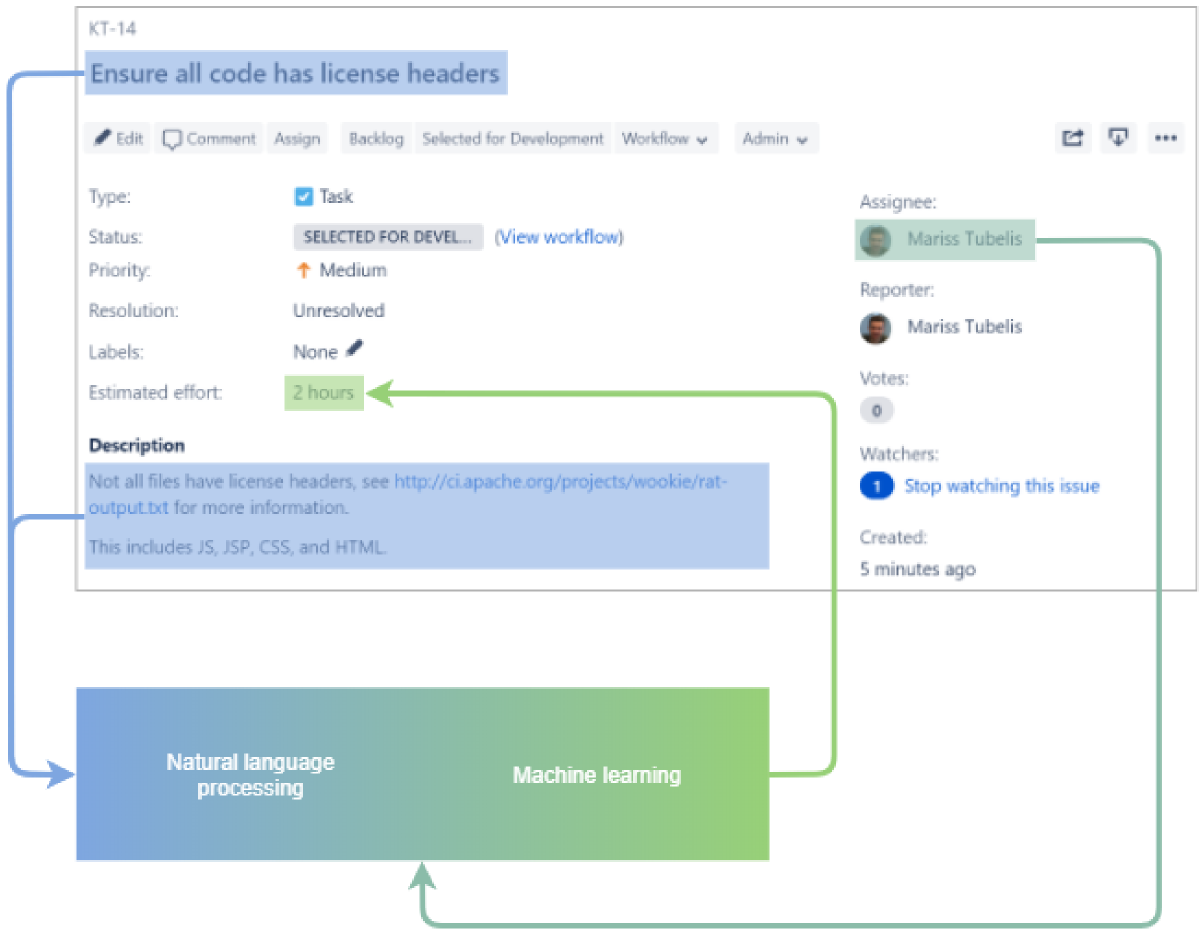
In the *first stage*, the tool is trained on 40 000 software engineering tasks with reported time spent from open source projects. These tasks account only for 3% of the total number of issues in the open source projects. In the *second stage*, the relationship between work description and time spent on a task is captured from commercial software projects under a strict confidentiality agreement. So far, two companies have agreed to participate in the project allowing access to 44 000 issues with time spent reported accounting for 66% of the total issues tracked in their JIRAs.

Confidentiality Rules

The model is trained at the participating company's office on a company-owned computer without Internet access according to a strict non-disclosure agreement. The student stores the machine learning model on a flash memory, which is handed over to the company for investigation after the training session. The company can then choose to withhold the flash memory if any rules are violated. The student is obliged to maintain secrecy for 5 years after the project completion date. The machine learning model exclusively captures the relationship between effort and its drivers in a form of numeric weights, ensuring that operational and business matters that for competitive reasons have importance for the company are not compromised by reverse engineering methods.

Technical Details

Although some of the technical details may change during the project, the main idea is to convert textual task representations to word vectors and then feed them to a recurrent neural network thus obtaining unadjusted effort estimates. Categorical features such as assignee's productivity may be used to enhance the estimation accuracy. A conceptual overview is depicted in the image below.



KT-14

Ensure all code has license headers

Type:	Task
Status:	SELECTED FOR DEVEL... (View workflow)
Priority:	Medium
Resolution:	Unresolved
Labels:	None
Estimated effort:	2 hours

Description

Not all files have license headers, see http://ci.apache.org/projects/wookie/rat-output.txt for more information.

This includes JS, JSP, CSS, and HTML.

Assignee:	Mariss Tubelis
Reporter:	Mariss Tubelis
Votes:	0
Watchers:	1 Stop watching this issue
Created:	5 minutes ago

Company Involvement

The tool is mainly trained on tasks written in English with logged time spent. The companies are provided with concise instructions adjusted to the systems they use prior the training session. The total expected amount of time spent on collaboration activities does not exceed 90 minutes, including the review of the confidentiality agreement and communication.

Appendix 2: Confidentiality Agreement with Collaborating Companies Translated from Latvian to English

Confidentiality Agreement

12 February 2018 Riga, Latvia

Mariss Tūbelis, personal identity number in Latvia .., personal identity number in Norway ... (hereinafter referred to as the Student), address ... and
... registered in Latvia with registration number ..., registered office ..., Riga, Latvia (hereinafter referred to as the Company),
which by virtue of the Articles of Association is represented by the member of its board ...,
hereinafter each of them separately referred to as the "Party", both of them collectively as the "Parties", shall enter into the following agreement (hereinafter referred to as the Agreement):

1. The Company allows for the Student to study the Correlation between the characteristics of the tasks related to the software development and the working time required for the task (Correlation), by means of the information available from the information systems used in the Company (Confidential Information), within the framework of the Master's thesis of the Student, in which JIRA and TFS Project Planning Systems Plug-in for Automated Labour Intensity Evaluation (Purpose) is developed in the Norwegian University of Science and Technology.
2. The Student undertakes to use the Confidential Information only for the Purpose referred to in Paragraph 1 of the Agreement.Use of the Confidential Information for other purposes is permitted only by virtue of a separate written agreement.
3. The Student undertakes not to disclose the Confidential Information to third parties.
4. The commitments referred to in Paragraphs 2 and 3 of the Agreement shall be effective [for 5 years] after the date of conclusion of the present Agreement.
5. The Company shall provide access and the Student shall study the Confidential Information (Information Analysis) according to the following provisions:
   - 5.1. The Student undertakes not to abandon the Confidential Information on data storage devices or cloud service environments outside the Company premises or which the Company does not use for data storage during the Information Analysis;
   - 5.2. The Information Analysis is carried out at the Company premises;
   - 5.3. The Student may bring into the Company premises and use in the Information Analysis an artificial intelligence tool, which is saved in flash memory and uses, inter alia, natural language processing and machine learning technologies;
   - 5.4. During the Information Analysis the Student saves in flash memory only and exclusively the explored Correlation in addition to artificial intellect tool;
   - 5.5. After the Information Analysis, the Student shall deliver the flash memory and access to the data stored therein to the Company which shall carry out the flash memory test within three working days;
   - 5.6. If the Company finds that the Student has violated the Agreement during the flash memory test, the Company shall retain the right to prevent the flash memory from returning to the Student;
   - 5.7. If the Company provides a computer for the Student for performance of the Information Analysis, the Student shall not be entitled to bring data storage and processing devices into the Company premises with the exception of the flash memory referred to in Paragraph 5.3:
   - 5.8. If the Company does not provide a computer for the Student for performance of the Information Analysis, the Students uses his own data storage and processing device in the Information Analysis, which is subject to the same rules as the rules referred to in Paragraphs 5.3 to 5.6 of the Agreement regarding use of the flash memory.

6. The commitments referred to in Paragraphs 2 and 3 of the Agreement shall apply to the entire Confidential Information, which the Company reveals to the Student during the Information Analysis or communications related thereto, while does not apply to:

6.1. The Correlation explored by the Student on the basis of the Confidential Information;

6.2. The Information which, after conclusion of the Agreement, becomes public domain (except if it has been disclosed in breach of the Agreement);

6.3. The Information already known to the Student and not subject to any non-disclosure obligations before the Company disclosed it to the Student;

6.4. Unclassified information distributed by the Company within the framework of its economic activity and which is available in public domain.

7. The Student in his Master's thesis may refer to the fact of the Information Analysis by reference to the name of the Company only in the case when at least one Company has been involved in the research.

8. The Student undertakes not to identify the Company, but to call it in another, fictional name when using the Correlation in JIRA and TFS Project Planning System Plug-in development and describing the effects of the Information Analysis carried out at the Company on the Master's thesis research results.

9. The Student undertakes to keep the Correlation in a form such as it is not possible to reproduce the Confidential Information by using reverse engineering techniques.

10. The conditions of the Agreement shall not apply to the cases where, in accordance with provisions of regulatory enactments the Student shall be under obligation to provide information subject to specific procedure and amount to the persons, authorities and institutions, which in accordance with the powers specified in regulatory enactments shall be entitled to request and to receive such information in performance of their functions.

11. The Parties may amend the Agreement with separate amendments in writing. Amendments to the Agreement shall constitute integral part of the Agreement.

12. Any amendments to the Arrangement shall enter into force only after they have been signed by both Parties and shall become an integral part of the Agreement. Oral amendments to the Agreement shall not be binding for the Parties.

13. If any paragraph of the Agreement is recognised as illegal or invalid, it shall not affect other provisions of the Agreement.

14. The Agreement has been drawn up and signed in duplicate, one original for each Party.

15. The Agreement shall be drawn up and interpreted in accordance with the regulatory enactments of the Republic of Latvia.

16. Any disputes arising from compliance with the Agreement or with the confidentiality obligation shall be resolved between the Parties before courts of general jurisdiction in accordance with the procedures specified in regulatory enactments of the Republic of Latvia.

17. The Agreement shall enter into force at the time of its signature.

Signature:	According to the Articles of Association ... representing, signed: ...
Mariss Tūbelis	…
___________________________	___________________________

Appendix 3: Publicly available JIRA repositories with at least 100 labeled datapoints

Name	Link	Labeled issues	Unlabeled issues	Labeling coverage
EXO	jira.exoplatform.org	9 733	26 581	26.80%
TDF	jira.talendforge.org	9 086	91 690	9.02%
GZL	gazelle.ihe.net/jira	7 049	6 268	52.93%
APC	issues.apache.org/jira	6 953	754 693	0.91%
JRA	jira.atlassian.com	5 815	219 405	2.58%
JBO	issues.jboss.org	3 929	296 486	1.31%
MRS	issues.openmrs.org	2 252	14 068	13.80%
EZZ	jira.ez.no	2 156	23 470	8.41%
PTH	jira.pentaho.com	2 072	37 665	5.21%
SPG	jira.spring.io	1 859	61 303	2.94%
MDB	mariadb.atlassian.net	1 847	6 757	21.47%
SNT	issues.sonatype.org	1 612	44 198	3.52%
NIH	tracker.nci.nih.gov	1 586	10 630	12.98%
OLS	openlmis.atlassian.net	1 510	3 128	32.58%
HIB	hibernate.atlassian.net	1 018	22 998	4.24%
SKP	jira.sakaiproject.org	548	39 454	1.37%
SEC	jira.secondlife.com	520	6 037	7.93%
MFG	mifosforge.jira.com	506	9 913	4.86%
ECS	ecosystem.atlassian.net	469	26 606	1.73%
OPC	opencast.jira.com	454	11 402	3.83%
HSC	ticket.hilscher.com	415	6154	6.3%
PCN	jira.percona.com	409	8 218	4.74%
MDL	tracker.moodle.org	270	82 628	0.33%
XWK	jira.xwiki.org	251	23 398	1.06%
KYL	kylo-io.atlassian.net	226	1 935	10.46%
DSP	jira.duraspace.org	155	11 230	1.36%
WSO	wso2.org/jira	144	81 360	0.18%
LNR	projects.linaro.org	140	1 018	12.09%
TPP	thepluginpeople.atlassian.net	133	2 594	4.88%
ONP	jira.onap.org	121	12 659	0.95%
ZNT	zanata.atlassian.net	119	2 215	5.10%
BQT	bugreports.qt.io	117	98 640	0.19%

Appendix 4: Label distribution in commercial datasets

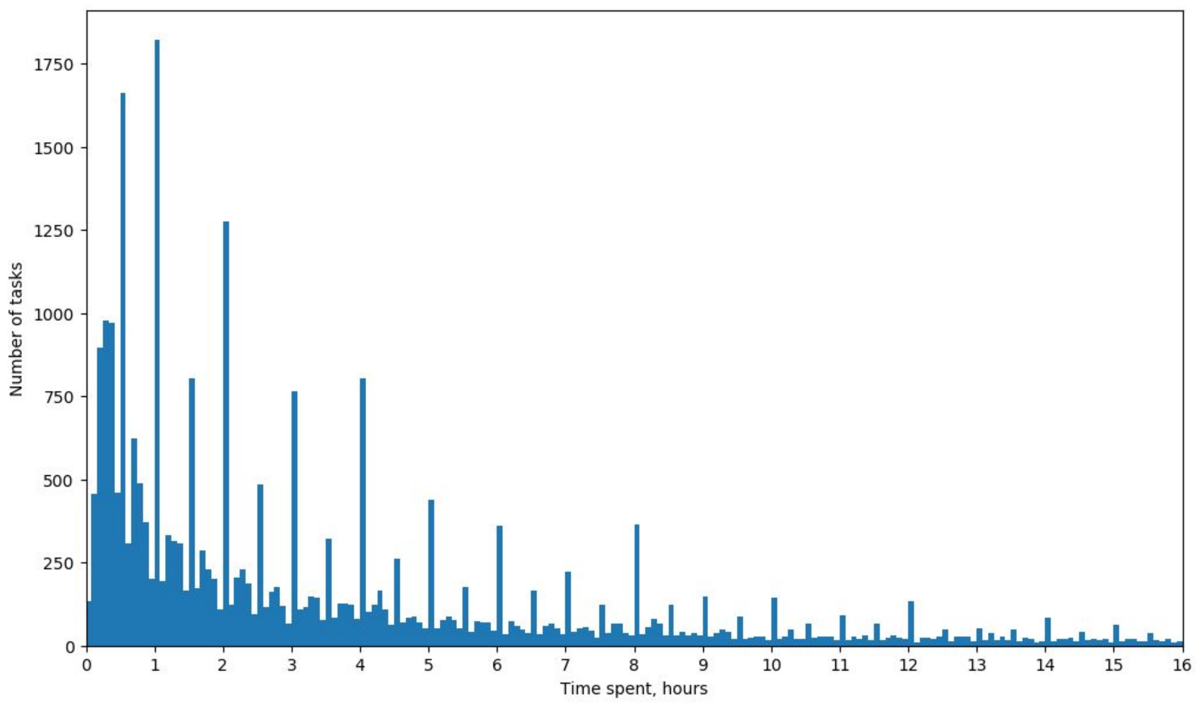


Image A Time spent on programming tasks up to 16 hours in the first commercial dataset

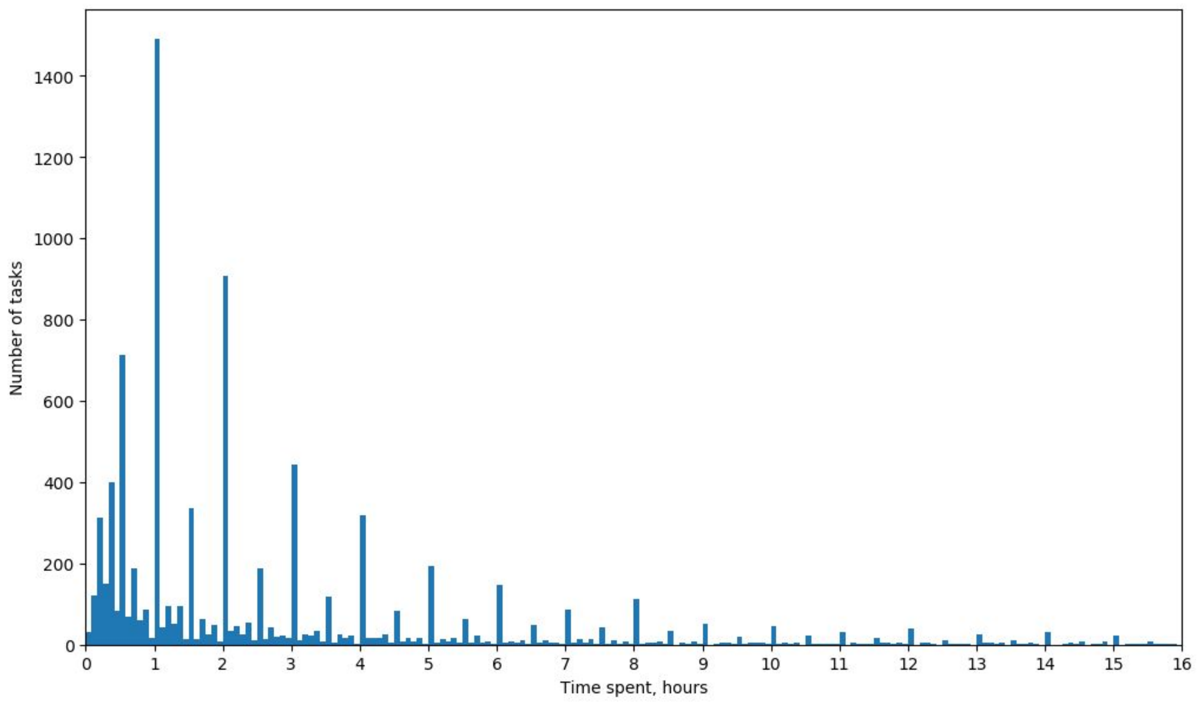


Image B Time spent on programming tasks up to 16 hours in the second commercial dataset

Appendix 5: Textual task descriptions length of labeled datapoints in commercial datasets

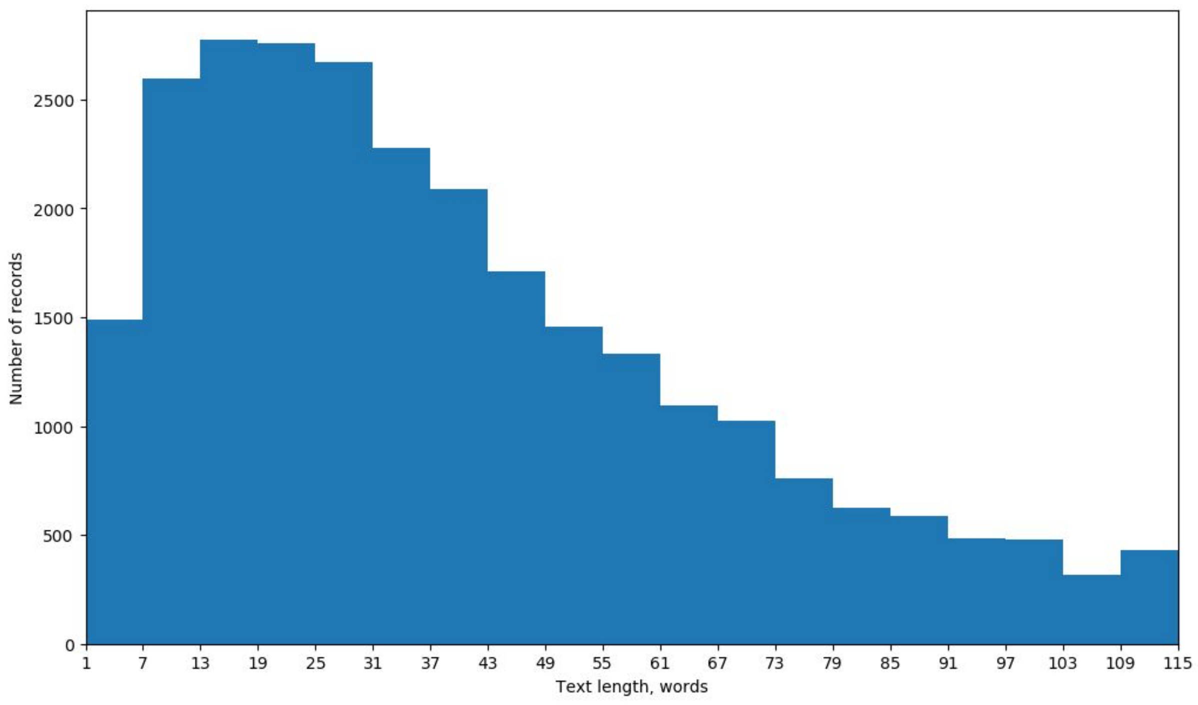


Image A Labeled datapoint text length in the first commercial dataset

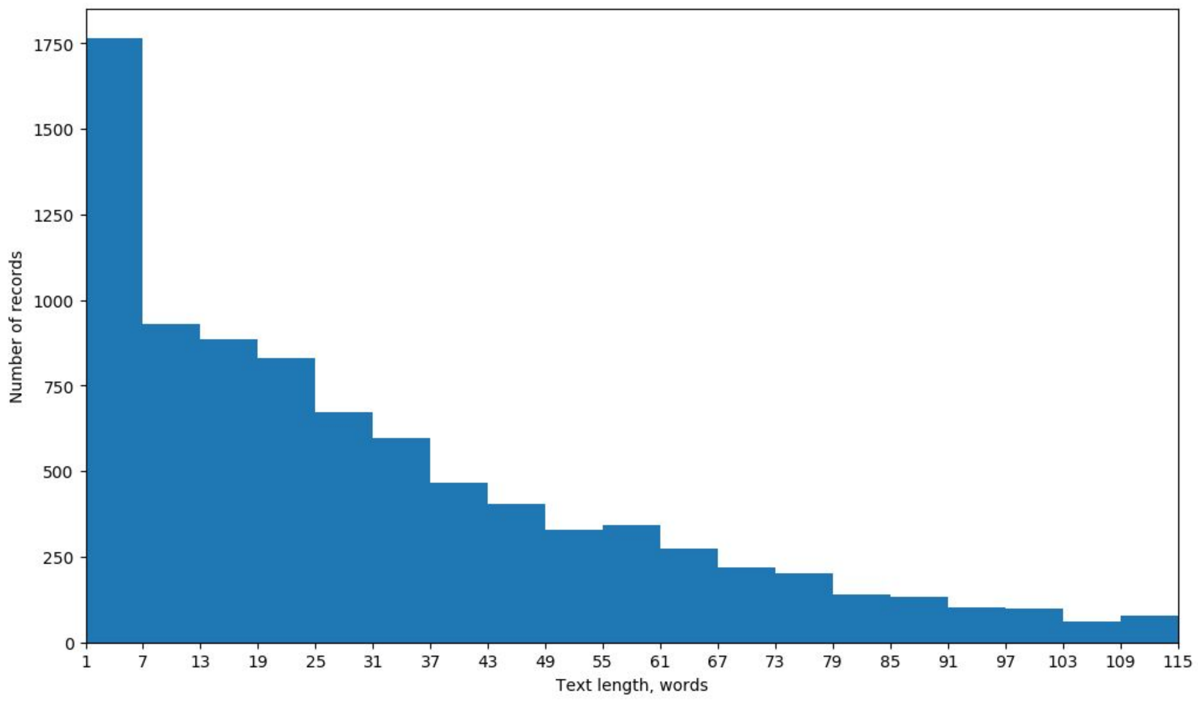
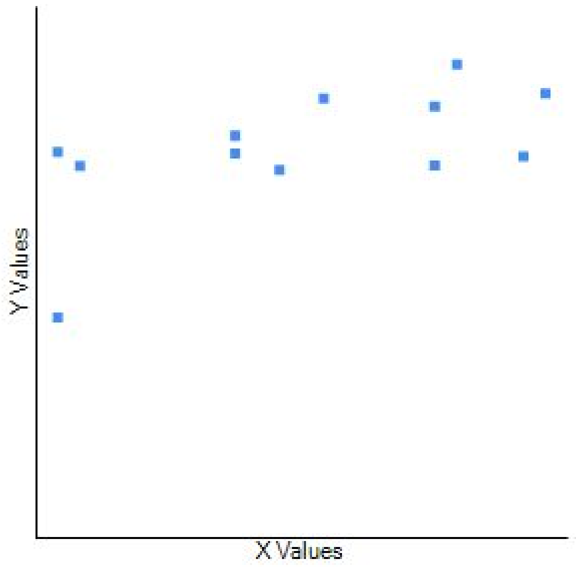


Image B Labeled datapoint text length in the first commercial dataset

Appendix 6: Correlation test between baseline differences and average model results

Dataset	Average result	Baseline		Baseline difference
		Testing	Validation	
all_1_1	7.28%	2.82	2.87	0.05
all_1_10	7.02%	2.77	2.83	0.06
all_1_20	6.94%	2.75	2.90	0.15
all_200_1	7.58%	2.79	2.92	0.13
all_200_10	8.28%	2.73	2.90	0.17
all_200_20	8.13%	2.76	2.98	0.22
all_500_1	7.25%	2.82	2.95	0.13
all_500_10	8.92%	2.75	2.98	0.23
all_500_20	8.37%	2.75	3.02	0.27
all_1000_1	7.20%	2.77	3.03	0.26
all_1000_10	7.03%	2.69	2.91	0.22
all_1000_20	4.17%	2.83	2.88	0.05

Pearson correlation coefficient between average results and baseline difference is 0.6066.



Y = average result, X = baseline difference