Studies in Informatics and Control, 26(2) 219-228, June 2017 ISSN: 1220-1766 eISSN: 1841-429X
https://doi.org/10.24846/v26i2y201710

Natural Language Processing and Machine Learning Methods for Software Development Effort Estimation

Vlad-Sebastian IONESCU[1], Horia DEMIAN[2] and Istvan-Gergely CZIBULA[1]

[1] Babeş-Bolyai University, 1, M. Kogălniceanu street, Cluj-Napoca, 400084, Romania,
{ivlad, istvanc}@cs.ubbcluj.ro
[2] University of Oradea, 1, Universității street, Oradea, Romania.
horia_demian@yahoo.com

Abstract: The growing complexity and number of software projects requires both increasingly more experienced developers, testers and other specialists as well as a larger number of persons to fill these roles. This leads to increased personnel and management costs and also makes effort and cost estimation at task and activity levels more difficult for software development companies. An automated solution for software development effort estimation based on text descriptions of tasks and activities, combined with available metrics, is introduced. A real world case study consisting of data from a software company whose activity spans a rich development spectrum is conducted. The results obtained are very encouraging and surpass the few similar approaches available in research literature.

Keywords: Software development effort estimation, Machine learning, Word embeddings, doc2vec, Support vector machines, Gaussian Naive Bayes.

1. Introduction

Software development effort estimation (SDEE) represents the action of estimating the time it will take for each part of a software system to be completed during the development phase of the product. Accurate estimates are important in order to properly plan the development process and allocate human resources accordingly.

An automated solution for the SDEE problem is introduced, consisting of a supervised machine learning framework that, after training, takes as input textual descriptions of required tasks and returns a numeric value representing an estimate of the effort required for completing those tasks. According to a literature review conducted in Section 3, the proposed approach is novel, with only one other approach even considering using textual descriptions of tasks in order to provide effort estimates. The results obtained are consistent and encouraging across a software company's entire projects base. The remainder of the paper is structured as follows. The motivation for this work is given in Section 2 Section 3 reviews several existing algorithmic approaches for effort estimation. Section 4 presents the fundamental concepts related to the machine learning models used in this paper. In Section 5, the introduced data sets and experimental methodology are presented. The case studies and the machine learning-based proposal for effort estimation are presented in Section 6, together with the experimental results which were obtained on real world data sets from a software development company. Section 7 analyses the obtained experimental results and compares them to existing similar work from the literature. The conclusions of the paper and directions for future research are outlined in Section 8.

2. Motivation

In an effort to mimic the real life effort estimation process, a natural language processing and machine learning based approach for the effort estimation problem is introduced. The working hypothesis is that, in a software project, for every resolved task, textual descriptions for the task can be found in the form of comments in the source repository (or associated with it), together with the actual time spent by developers on the task, by analysing the logs. Using machine learning, relations can be discovered between the textual description and the needed time. Intuitively, this corresponds to the domain knowledge and experience factors in the actual effort estimation done by a software developer.

In most Agile development methodologies, the smallest unit to estimate is a **task**. A task has a

description, is usually linked to a user story, a feature request or a usage scenario [4]. Software developers are asked to give an estimate based on this textual information. At first, the only input for the estimation problem is some textual representation of the problem to be solved.

Automated software effort estimation can be used as a means to verify and correct actual estimates made by developers. By comparing the estimation (or reported effective time) given by the developer with the predicted one, project managers can identify problematic tasks or incorrect reporting.

While this proposal is complex, the complexity lies in its research and development, not in its application. Applying it is as easy as applying any well-known machine learning methodology in any other field, and it saves developers time otherwise spent on providing (sometimes daily) estimates.

Since accurately estimating the software development effort is a difficult and important task, for which a lot of human estimation methodologies, as well as some automated methodologies, exist, this paper considers machine learning based regression models to be appropriate for providing estimates. In order to solve the issue of needing project metrics and human input for SDEE systems, the goal is to feed machine learning models with only the textual descriptions of the tasks that need solving.

3. Software Development Effort Estimation

This section gives an overview of **Planning Poker**, a popular human SDEE methodology used in Agile environments, and the existing automated approaches to SDEE together with a literature review of their results.

3.1. Planning Poker

Planning Poker is a consensus-based method for estimating the effort required to solve programming tasks. Its main goal is to force developers to think independently and reach a proper consensus regarding the effort required, without one person influencing the rest [4].

Typical Planning Poker uses a deck of cards with Fibonacci numbers on them, starting from 0 up until 89. These represent effort, measured in any unit convened upon, such as hours [4].

First, the team can discuss the requirements in order to clarify any uncertainties, without mentioning any estimates, in order to avoid influencing each other. Then, they lay a card face down. Once everyone has decided on a card, they each turn them up at the same time.

The process continues until a consensus is reached.

An advantage of Planning Poker over more ad-hoc methods is that it can reduce personal biases and it forces developers to be able to properly defend their choice. An important disadvantage is that it takes more time due to the multiple rounds and people involved.

Like in most complex fields, there is no silver bullet: planning poker is a tried and true method that adapts well in many situations, but steals important development time. The goal of this research, and of most research in automating SDEE, is not yet to replace humans completely. The current state of the art in machine learning cannot do that currently. The goal of this proposal is to provide alternatives for cases in which some stated assumptions hold and the drawbacks of other methods outweigh our own method's drawbacks.

3.2. Algorithmic approaches to software development effort estimation

Most computational approaches to SDEE rely on a mathematical formula that considers certain project metrics and on domain knowledge. This makes them unlikely to perform well on a large variety of projects, and their use is mostly avoided in practice. The below are generally called **parametric models**.

COCOMO

COCOMO [3] starts by dividing projects into three types. Then, COCOMO provides three formulas for effort, one for each type of project.

An important disadvantage of the model is that it tries to accommodate the existence of a lot of important project factors into its estimations by providing either tables of values or human estimates. Tables are not robust, cannot easily be adapted to one's own situation and are subjective. The necessity of human estimates and evaluations means that it does not provide a fully automated solution.

Putnam model

The Putnam model [12] bases its estimations on similar formulas. It is known that the method is

sensitive to its parameters, which must be estimated by human factors.

An advantage of the Putnam model is its calibration simplicity, however it still suffers from the need of human estimations, and can be inaccurate in practice.

Many other approaches similar to COCOMO and the Putnam model exist in the literature, however they mostly rely on some fixed equations involving a number of subjective, user-inputted parameters, which reduces their robustness and resilience to human error.

3.3. Accuracy of estimates

The most widespread accuracy metric for effort estimation systems is the **Mean Magnitude of Relative Error** (MMRE), shown in Formula (1), where n is the number of tasks estimated, EA_i is the actual effort for task i and EE_i is the estimated effort for task i. The objective is to minimize the MMRE.

$$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{|EA_i - EE_i|}{EA_i} \quad (1)$$

Note that, the equation in Formula (1) is sometimes multiplied by 100, in order to express the estimation error as a percentage.

3.4. Related work on algorithmic effort estimation

Consider the SDEE literature divided in three categories, with regards to how close the used techniques are to our own approaches.

Classical parametric models

The first category consists of the classical frameworks discussed in the previous section, such as COCOMO. There are many studies that apply these frameworks to various projects, usually private ones on which, unfortunately, none of our own methods can be applied in order to provide a direct comparison, as the project data is not publicly available.

In the study at [1], Basha and Ponnurangamthe apply the COCOMO, SEER, COSEEKMO, REVIC, SASET, Cost Model, SLIM, FP, Estimac and Cosmic frameworks to a set of applications of various types, such as Flight Software and Business Applications, obtaining MMRE values between 0.373% and 771.87%. The authors conclude that there is no one best framework and that they are all very sensitive to the input data, the application type and the various abilities of the development team.

Popovi'c and Boji'c analyse in [11] 94 projects developed between 2004 and 2010 by a single company. These are mostly Microsoft .Net Web projects with a lot of available metrics and documentation. The obtained MMRE values are between 10\% and 46\%, using linear and non-linear models with various metrics and phases at which effort is estimated. Once again, the data set used is not publicly available.

A set of open source projects is experimented on by Toka in [16] using COCOMO II, SEER-SEM, SLIM and TruePlanning. The MMRE values range from 34% using TruePlanning to 74% using COCOMO II.

In a recent literature survey on Software Effort Estimation Models, Tharwon presents in [15] an overview of experimental research that uses the Function Point Analysis (FPA), Use Case Point Analysis (UCPA) and COCOMO models.

The MMRE value obtained by the FPA model in the surveyed case studies is at least 13.8% and at most 1624.31%, with an average across the case studies of 90.38%. Considering UCPA, the minimum MMRE value of four surveyed experimental papers is 27.30%, the maximum 88.01% and the average is 39.11%. Considering COCOMO, the average is 281.61%.

A comparison that also includes human estimates, provided through planning poker or by an expert, is performed by Usman et al. in [17]. On the considered projects, it found that planning poker obtains a MMRE of 48%, UCPA methods obtain MMRE values between 2% and 11% and expert judgments between 28% and 38%.

As confirmed by the literature, the vast majority of the time, parametric models do not provide useful effort estimates.

Machine learning models using software metrics

Machine learning models using software metrics are understood to be frameworks such as COCOMO that are used together with more advanced, machine learning-oriented elements, such as fuzzy logic, neural networks, Bayesian statistics and the like. Approaches that use pure machine learning algorithms applied exclusively on various project metrics and indicators are also considered.

A Neuro-Fuzzy approach is used by Du et al. in conjunction with SEER-SEM in [6] in order to obtain lower MMRE values on four case

studies consisting of COCOMO-specific data. The obtained MMRE values using the classical SEER-SEM approach are between \$42.05\%\$ and 84.39%. Using the Neuro-Fuzzy enhancement, they are between 29.01% and 69.05%, which is a significant improvement.

Han et al. apply in [9] a larger set of machine learning algorithms: linear regression, neural networks, M5P tree learning, Sequential Minimal Optimization, Gaussian Process, Least Median Squares and REPtree. The study is conducted on 59 projects having between 6 and 28 developers and between 3 and 320 KLOC. The obtained MMRE values are between 87.5% for the Linear Regression approach and 95.1% for the Gaussian Process model.

Bayesian networks, Regression trees, Backward elimination and Stepwise selection are applied on various metrics from two software project data sets by van Koten and Grayin [19]. The best obtained MMRE is 97.2% on one of the projects, using Bayesian networks, and 0.392%, using Stepwise selection, on the other project.

In a literature review of machine learning models applied to the SDEE problem [20], Wen et al. show that MMRE values fluctuate a lot between different projects as well as different learning algorithms. For example, for Case Based Reasoning, the survey found experiments with MMRE values between 13.55% and 143%. Similar ranges were found for Artificial Neural Networks, Decision Trees, Bayesian Networks, Support Vector Regression and Gaussian Processes.

In [17], Usman et al. obtain MMRE values between 66% and 90% using linear regression. Using Radial Basis Function networks, MMRE values between 6% and 90% are obtained.

According to our literature review, machine learning models applied on software metrics provide better estimates than pure parametric models. The MMRE values are also less spread out between different data sets, which makes machine learning models more reliable and predictable from an accuracy point of view.

However, a remaining drawback of these approaches is the need for project software metrics, which are not always available or would take substantial effort to collect properly. Sometimes, various parameters must still be inputted by the developers, which takes about as much time as it would take developers to provide their own estimates.

Machine learning models using text processing

To the best of our knowledge, the thesis by Sapre in [14] is the only other research that approaches the SDEE problem by inputting task descriptions directly to ML learning pipelines. It uses a bag of words approach on keywords extracted from Agile story cards, which it then feeds to multiple learning models. Experiments are conducted both with the Planning Poker estimates included in the actual learning part of the pipeline and without. The author reports 106.81% MMRE for Planning Poker estimates, and 92.32% MMRE using J48 (which outperforms the other models) with the Planning Poker estimates excluded from the learning stage. Including the Planning Poker estimates leads to slightly better results, although not enough so as to not defeat the purpose of an automatic approach.

The approach classifies instances into classes representing Fibonacci numbers, in the same way that Planning Poker estimates are provided.

4. Fundamentals of the machine learning elements used

This section presents the fundamental of the machine learning elements used throughout our experiments: **term frequency-inverse document frequency** (TF-IDF), **distributed representations of documents** (doc2vec) and **support vector regression** (SVR). Gaussian Naive Bayes (GNB) classification is also used in order to replicate an approach from the literature.

4.1. Term frequency-inverse document frequency (TF-IDF)

TF-IDF is a weighting scheme for terms in a text corpus. It represents the multiplication between the term frequency statistic, that counts how many times a term appears in a document, and the inverse document frequency statistic, that is the inverse fraction of the documents that contain the term.

The TF-IDF process is usually the first step of a text processing machine learning pipeline. Its results are then fed to classifiers and regressors.

4.2. Distributed representations of documents (doc2vec)

Models such as word2vec [7] and doc2vec [8] address a key weakness of bag of words models

like TF-IDF: that words lose their semantics in the process. For example, in TF-IDF, there is no necessarily stronger relationship between the words “Paris” and “London” than between the words “Car” and “Skyscraper”. In distributed vector representation models, the model would learn, from a large enough corpus, that “Paris” and “London” are both capital cities, and their vectors would be closer together than those of words with less meaning in common.

This is achieved in word2vec by training a model to predict a word given a context, which is a set of words around it.

As shown in [19], this can be extended to documents as well, allowing us to obtain vectors for entire documents and to infer new vectors for unseen documents.

Because semantics are kept, feeding these vectors into classifiers and regressors, in a similar manner as the TF-IDF vectors are used, leads to better results for some tasks. Doc2vec vectors are employed for the same purpose.

Doc2vec vectors can be combined with TF-IDF vectors by multiplying the two, thus potentially keeping the information provided by both approaches. Our experiments consider this case as well.

4.3. Support vector regression

Cortes and Vapnik originally developed **support vector machines** for supervised classification [5], but they have also been successfully applied in regression. The regression method is known as ε-support vector regression (SVR), since an extra hyperparameter ε is used for controlling the algorithm's error level.

4.4. Gaussian Naive Bayes

The Gaussian Naive Bayes (GNB) is a classification algorithm that assumes that the values in each class follow a Gaussian distribution. This allows the algorithm to function without having to discretize the features.

First, the algorithm computes μ_c and σ_c, representing the mean and variance of all instances in class c. When having to classify a new instance, the algorithm uses the Gaussian distribution parameterized by μ_c and σ_c to find the probability of the instance belonging to class c. By not having to discretize the feature values, which would be required in order to apply the classical Naive Bayes algorithm, GNB makes use of all the available information in the data.

5. Data sets and methodology

In this section, our data sets and experimental methodology are presented.

5.1. Data sets

Description of data sets

The data sets consist of a templated data set (i.e. a data set following a given template, described below), which is called T, and eight other data sets, which are referred to as d_1, d_2, ..., d_8. All data sets are provided by a software company that deals with software development and general IT maintenance work. The software development activities are desktop and web-related using Microsoft .NET technologies and the maintenance work consists of network management, printers servicing and other such work.

The T data set consists of tasks that are described by team members in a certain format that is meant to help learning algorithms infer estimates more accurately. For our T set, all of the following holds true:

- A task has one or more **actions** that have to be performed with the goal of completing the task.
- A task can take one or more days to complete.
- An action refers to an indivisible set of development activities that have to be performed during a single work day.
- One or more actions, done by one or more developers, can be necessary to complete a task.
- Each instance describes a task and its associated actions.
- The content of a task represents a development goal.
- The content of an activity represents what was done in order to achieve the goal (reinstalling a program, installing some hardware, restoring a backup etc.).

Only development (programming) tasks are described.

In T, an instance consists of text representing:

1. For the task:

- The **Interface** worked on: the name of a database table, class, source file etc.
- The **Complexity**: as estimated by the project manager, an integer between 1 (trivial) and 5 (very difficult).
- The **Number of entities**: how many entities the change will affect, usually an objective measure.
- The **Estimation**: a very rough human estimate, in minutes.
- The **Functionality**: a short textual description of the goal.

2. For each action:

- The **Description**: a short textual description of the action.
- The **Type**: one of "Creation" or "Change", representing if something new was added to the project (a file or database table, for example), or if an existing entity was somehow changed.

3. The last number represents the actual time in minutes it took to complete the task.

For example, an instance of T can look like this:

Interface: Catalog
Complexity: 4
NumEntities: 2
Estimation: 100
Functionality: Add a new admin only field for the internal product rating
Description: Insert new columns
Type: Change
Description: Make new column admin only
Type: Change
150

This describes a task with two actions, since "Description" and "Type", which are particular to actions, appear twice. It took 150 minutes to complete the task.

The d_1 through d_8 data sets have a simpler format and they refer to non-programming tasks. The following is what each instance of a d_i data set contains:

- A textual description of the task, similar to the "Functionality" field of the instances of T.
- A number representing the count of physical systems the client has that are managed by the company.
- Another number representing the licensed software count that the client has and that must be managed by the company.
- A final number representing the actual time in minutes it took to complete the task.

Table 1 presents the number of instances in each data set, along with a short presentation of each data set's contents.

Each d_i data set contains instances until the same time as those for T were collected.

Note that our data sets are diverse: T was collected over a relatively short period of time with the express purpose of being adequate for the SDEE problem, while the others represent a simple, ad-hoc, internal tracking of the company's business. Moreover, the d_i data sets were collected over a period of a few years, with more developers introducing data in their own particular styles, since there was no guideline for how descriptions should be written.

Table 1. Number of instances and short presentations for each data set.

Data set	Number of instances	Short presentation
T	203	The templated data set, containing data collected over a five months period.
d_1	147	Contains data referring to network administration tasks, since the company started tracking them (a few years).
d_2	1756	Contains data referring to financial software activities, such as receipts, billings etc., since the company started tracking them (a few years).
d_3	138	Contains various maintenance activities.
$d_4 – d_7$	318, 564, 220, 194	Same as above.
d_8	862	More general hardware maintenance tasks.

For both the T and d_i data sets, the descriptions are very short, usually not containing more than 10 words.

Visualization of data sets

In order to get a better understanding of the complexity and difficulty of the problem at hand, consider a visualization of the data sets in two dimensions. This is achieved this by applying t-SNE [18] to the values returned by applying either TF-IDF, doc2vec or TF-IDF xdoc2vec (with and without parsing) on all instances of a data set. By visualizing the data in two dimensional space, intuition is gained about how well learning is likely to work on a certain data set.

For this purpose, only the T data set is considered. Figure 1 shows visualizations for the T data set using TF-IDF with parsing (using doc2vec is similar). It can be seen that this is a difficult problem, since simple linear regression does a very poor job of fitting the reduced data sets.

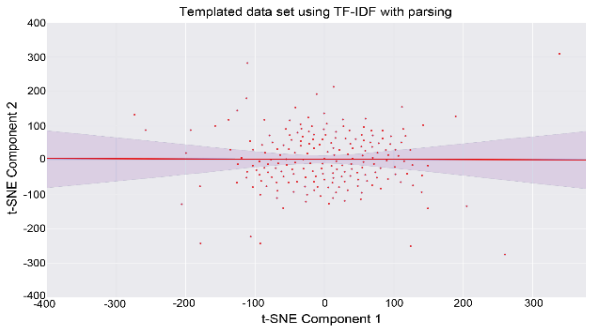

Figure 1. Visualization of TF-IDF transformer reduced to two dimensions on the *T* data set, with initial preprocessing (parsing).

5.2. Experimental methodology

The experiments consist of a machine learning pipeline with multiple steps, a hyperparameter search and a final model evaluation.

Before the learning starts, the data set is first of all randomly shuffled.

The machine learning pipeline

All experiments start with and without a text preprocessing step that differs based on the type of data set: T or a d_i set. Then, they proceed in the same manner.

For the T set, this preprocessing step consists of transforming the text of each instance as follows. Since the fields for each action can appear multiple times, they are concatenated such that the keywords **Description**, **Type** etc. only appear a single type, followed by the contents of all of them. The numeric contents of the **Complexity**, **Number of entities** and **Estimation** numeric fields are also copied into a separate **numerics** vector that will be carried over to the next stages of the pipeline together with the preprocessed text. Note that the preprocessed text still contains these numeric fields.

For the d_i data sets, the preprocessing consists of simply copying the numeric fields into the separate **numerics** vector.

Experiments are performed with and without the initial preprocessing step. Without it, the raw text data, as described in the previous subsection, is fed to the next stages.

The next part of the learning pipeline is the transformation of the text into a vector model. Experiments are performed using TF-IDF and doc2vec. In case the first preprocessing step is applied, the vector model is fed to the next part of the pipeline concatenated with the numerics vector and scaled to zero mean and unit variance.

The final part of the pipeline is using an actual learning model to learn relations between the text and the real completion times.

The proposed method relies on the SVR algorithm. Tests are also run with GNB in order to compare the proposed methodology with the one used in [14] where the authors employ abag of words approach that results in discrete features. Since TF-IDF and doc2vec do not produce discrete features, GNB was chosen so as not to lose information by discretization. The authors employ classification into Fibonacci classes, as used in Planning Poker. Each training instance is put in the class corresponding to the Fibonacci number closest to its actual effort. MMRE values are computed by using the Fibonacci numbers associated with each class. The same is done on the introduced data sets.

Hyperparameter search

The pipeline involves many hyperparameters for the TF-IDF or doc2vec stages and for the SVR stage. There are no hyperparameters for GNB. Hyperparameters need proper values in order to obtain good results. Since there are so many, it is impossible to run a full grid search over them, so a random search is used, which has been shown to provide good results in general [2].

For the hyperparameter search, in the case of TF-IDF, values are sampled for 11 TF-IDF-specific hyperparameters, such as the level of ngrams (character or word), the ngram range, the maximum number of features to keep, the normalization method etc., either from discrete sets of likely to perform well values or from uniform distributions over known good ranges. In the case of doc2vec, 11 doc2vec-specific hyperparameters are sampled, in similar manner.

For the SVR model, hyperparameters such as C and other SVR-specific ones are sampled from the same type of distributions. Only the linear kernel is considered in our experiments, having found that others take much longer to evaluate and provide almost no improvements.

The hyperparameter search runs for 5000 iterations on each experiment, using **10-fold cross-validation** (10CV) to evaluate each configuration.

Model evaluation

Once the random hyperparameter search completes, the best result it has found for some configuration of hyperparameters, according to our sampling sets and distributions, is reported. The best ones are applied to a new pipeline, the data is reshuffled and the pipeline is evaluated on the data set again using 10CV. These are the reported results of the paper.

6. Experimental results

In this section, the experimental results on the real world data for SDEE, described in Section 5.1, are reported. The experiments are divided by the **text representation** method used (TF-IDF, doc2vec and TF-IDF x doc2vec) and within each representation method by whether or not the initial text preprocessing was used.

For each experiment, average MMRE on the test folds within 10 fold cross validation is reported. The scikit-learn machine learning library is used for experiments [10].

Table 8 shows that TF-IDF followed by SVR provides the best results on all but one data set. When using GNB, doc2vec and TF-IDF x doc2vec lead to better results most of the time, without surpassing the regression approach however. This suggests that doc2vec might be better for classification problems than for regression problems, at least in the context of SDEE. The training is also faster with TF-IDF. While doc2vec vectors store more data about the semantics of the documents, this did not improve our regression results. Similar observations hold when not using initial text preprocessing.

Note that, generally the best results were obtained on the T data set, which was specifically built with care to how the tasks are described, in order to help our algorithms perform better. This was successful, and shows that better written task descriptions can help improve MMRE results.

Table 8. Results using each text vectorizer with the initial text preprocessing. The best SVR and GNB results are highlighted across the three different text vectorizers.

Data set	Set size	TF-IDF		doc2vec		TF-IDF x doc2vec	
		SVR	GNB	SVR	GNB	SVR	GNB
T	203	0.53	0.668	0.589	0.592	0.565	0.727
d_1	147	0.593	0.705	0.683	0.731	0.637	0.716
d_2	1756	0.641	0.743	0.657	0.713	0.66	0.695
d_3	138	0.588	1.262	0.618	0.981	0.611	1.463
d_4	318	0.571	0.674	0.62	0.666	0.606	0.614
d_5	564	0.594	0.849	0.606	0.755	0.597	0.925
d_6	220	0.587	0.973	0.623	0.748	0.621	1.128
d_7	194	0.597	1.033	0.576	0.759	0.618	1.045
d_8	862	0.643	0.707	0.662	0.681	0.663	0.712

For both methods, there aren't big differences between the data sets. This shows that our machine learning approach to SDEE is robust and is likely to perform well on various data set.

7. Discussion and comparison to related work

Considering the results found in the literature and presented in Section 3.3, the results obtained on our data sets are better than most of the results found for this problem. Table 9 presents a comparison to the related work reviewed in Section 3.3.The most relevant comparison is with [14], due to the fact that it also uses raw text data for the experiments. Observe that, considering this related work, the proposed approach obtains significantly better results with both regression and classification. Moreover, the regression results on the introduced data sets are considerably better than the classification results, which indicates that regression could be the better choice for the SDEE problem.

8. Conclusions and future work

We have shown that using text vectorization methods such as TF-IDF and doc2vec, together with regression algorithms, can obtain better results for the SDEE problem than classical parametric models such as COCOMO.

Table 9. Comparison to related work. The related works with higher average MMRE values than our best result on the T data set are marked in green. Those for which we do better than the upper bound on the T set are marked yellow.

Related work	MMRE
[6]	0.373% -771.87%
[11]	10% - 46%
[16]	30% - 74%
[15]	13.8% - 1624.31%, 90.38% average.
[15]	27.30% - 88.01%, 39.11% average
[17]	48% for planning poker, 2% - 11% for UCPA, 28% - 38% for human estimates.
[6]	29.01% - 69.05%
[9]	87.5% - 95.1%
[19]	97.2% on one of the projects and 0.392% on the other.
[20]	13.55% - 143%
[17]	66% - 90% for linear regression, 6% and 90% for RBF networks.
[14]	92.32%

Our method is also, as far as we know, one of only two that uses text data for providing estimates. It is the only one that uses modern machine learning algorithms and regression in order to do this.

We are confident that better structured text can significantly reduce the MMRE values, as indicated by the generally lower errors on the T data set, which has a decent structure compared to the others. More data would be needed in order to properly assess this, however.

In the same way, our experiments also suggest the following:

- Regression approaches perform better for SDEE than classification into Planning Poker Fibonacci classes.
- Preprocessing the initial text using basic parsing strategies and extraction of numeric values helps obtain better results, especially when using regression.
- Basic TF-IDF vectorization leads to better results than more advanced methods, such as doc2vec, at least for regression. For classification, doc2vec and TF-IDF x doc2vec obtain slightly better results.

In the future, we plan to gather more data from more companies, in order to better determine which models work better and in which cases. We also plan to incorporate metrics in our data sets, so as to make use of the information they provide together with the information the textual description of a task provides.

Acknowledgments

This work was supported by a grant of the Romanian National Authority for Scientific Research and Innovation, CNCS--UEFISCDI, project number PN-II-RU-TE-2014-4-0082.

REFERENCES

1. Basha, M. S. S. & Ponnurangam, D. (2010). Analysis of Empirical Software Effort Estimation Models, *CoRR,* abs/1004.1239, URL: http://arxiv.org/abs/1004.1239.
2. Bergstra, J. & Bengio, Y. (2012). Random Search for Hyper-parameter Optimization, *J. Mach. Learn. Res.,* 13, 281-305, ISSN: 1532-4435.

URL:http://dl.acm.org/citation.cfm?id=2188385.2188395.

3. Boehm, B. W. et al. (2010). Software Cost Estimation with Cocomo II with Cdrom. 1st. ed., *Upper Saddle River, NJ, USA: Prentice Hall PTR*. ISBN: 0130266922.
4. Cohn, M. (2005). Agile Estimating and Planning, *Upper Saddle River, NJ, USA: Prentice Hall PTR*, ISBN: 0131479415.
5. Cortes, C. & Vapnik, V. (1995). Support-vector networks, *Machine Learning*, 20, 273-297. ISSN: 0885-6125.
6. Du, W. L.; Ho, D. & Capretz, L. F. A. (2015). Neuro-Fuzzy Model with SEER-SEM for Software Effort Estimation, *CoRR*, abs/1508.00032. URL: http://arxiv.org/abs/1508.00032.
7. Le, Q. V. & Mikolov, T. (2014). Distributed Representations of Sentences and Documents, *CoRR*, abs/1405.4053. URL: http://arxiv.org/abs/1405.4053.
8. Mikolov, T. et al. (2013). Distributed Representations of Words and Phrases and their Compositionality, *CoRR*, abs/1310.4546. URL: http://arxiv.org/abs/1310.4546.
9. Han, W. et al. (2015). Comparison of Machine Learning Algorithms for Software Project Time Prediction. *International Journal of Multimedia and Ubiquitous Engineering*, 10, 1-8. URL: http://dx.doi.org/10.14257/ijmue.2015.10.9.01.
10. Pedregosa, F. et al. (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, 12, 2825-2830.
11. Popović, J. A. B. D. A. (2012). Comparative Evaluation of Effort Estimation Methods in the Software Life Cycle, *Computer Science and Information Systems*, 455-484. URL: http://eudml.org/doc/253105.
12. Putnam, L. H. & Myers, W. (2003). Five Core Metrics: Intelligence Behind
13. Successful Software Management, *New York, NY, USA: Dorset House Publishing Co., Inc.* ISBN: 0932633552.
14. Rehurek, R. & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45-50). URL: http://is.muni.cz/publication/884893/en
15. Sapre, A. V. (2012). Feasibility of Automated Estimation of Software Development Effort in Agile Environments, *Ph.D. Thesis at The Ohio State University*.
16. Tharwon, A. (2016). A Literature Survey on the Accuracy of Software Effort Estimation Models. In *Proceedings of the International Multi Conference of Engineers and Computer Scientists*.
17. Toka, D. (2013). Accuracy of Contemporary Parametric Software Estimation Models: A Comparative Analysis. In *Proceeding of the 39th Euromicro Conference Series on Software Engineering and Advanced Applications* (pp. 313-316).
18. Usman, M. et al. (2014). Effort Estimation in Agile Software Development: A Systematic Literature Review. In *Proceedings ofthe 10th International Conference on Predictive Models in Software Engineering* (pp. 82-91).
19. Van Der Maaten, L. J. P. & Hinton, G. E. (2008). Visualizing High-Dimensional Data Using t-SNE, *Journal of Machine Learning Research*, 9, 2579-2605.
20. Van Koten, C. & Gray, A. R. (2006) An Application of Bayesian Network for Predicting Object-oriented Software Maintainability, *Inf. Software Technol., Newton*, 48, pp. 59-67, ISSN: 0950-5849. URL: http://dx.doi.org/10.1016/j.infsof.2005.03.002.
21. Wen, J. et al. (2012). Systematic Literature Review of Machine Learning Based Software Development Effort Estimation Models, *Inf. Software Technol., Newton*, 54, 41-59, ISSN: 0950-5849. URL: http://dx.doi.org/10.1016/j.infsof.2011.09.002.

