Effort estimation via text classification and autoencoders

Rodrigo G. F. Soares
Department of Statistics and Informatics
Federal Rural University of Pernambuco, Recife, Brazil
Email: rodrigo.gfsoares@ufrpe.br

***Abstract*—The estimation of the effort required for the production of a software or the correction of a software issue is an important activity in software development methodologies. Such estimates are essential to the delivery of high-quality products within a time frame and budget. Intuitively, these estimates should be produced as early as possible in the development process in order to improve planning. Issue reports are the earliest documents available for the correction of a software. We tackle effort estimation in early stages of agile software development as text classification of issue reports. This task consists of assigning categories to such documents, often written in natural language and programming code. Typically, text classification is performed with feature extraction techniques that produce informative features for a text classifier. Autoencoders generate input representations in increasing levels of abstraction, which may represent meaningful semantics in issue reports. We propose a study on the effectiveness of autoencoders for estimating the effort required for the agile correction of software functionalities in the context of real-world open-source projects. Our experiments provided significant evidences that autoencoders are able to encode noisy documents and provide informative features to a text classifier and improve its generalization.**

I. Introduction

With the increasing number of available documents, the importance of organizing and mining such sources of information becomes critical for many activities that depend on knowledge discovery. Acquiring high-quality information from such a plethora of documents is the main goal of text mining [1]. It involves text classification, clustering, concept extraction, document summarization, among others [2].

Categorizing documents in natural language has been widely studied due to its importance in various applications, such as document organization, news filtering, opinion mining and spam filtering [2]. It consists of automatically assigning labels to texts, often via Machine Learning (ML) algorithms [3]. Text data is typically unstructured and possesses large vocabularies, which leads to challenges in finding useful representations of texts for the training of supervised classifiers.

Text data is often transformed into vector spaces based on term frequencies. Term-based approaches produce high-dimensional feature spaces [4]. Such a dimensionality is one of the main problems in text classification, as computational cost becomes increasingly onerous [2]. Sparsity increases with the size of the vocabulary of the problem [5]. Text classifiers also should handle noise in texts: misspelled words, redundancies, grammar errors, etc. Such noise can generate irrelevant and misleading features, which might have great impact on text classification generalization [6]. For example, in agile software development documents, such text issues are common [7], [8], [6]. The use of feature extraction techniques in these documents can tackle these problems and reveal important information for the automatic categorization of such texts.

Delivering high quality products within a time frame and budget are objectives of software development processes [9], [10]. Project management can prevent failures in achieving these goals throughout the software development [7]. The planning of the software development process often uses estimates of effort that will be demanded by its production [11], [12]. Estimates help the project manager in the correct allocation of resources, such as time and team members. If these resources are underestimated, there may be increased costs during development and delays in deliveries. In case of overestimation, the institution may lose competitiveness. Thus, estimates are fundamental in the early stages of the software life cycle and incorrect estimates can ultimately cause the failure of a project [13], [3]. Resolving software issues also requires planning and can be considered as software development [14]. It also uses estimates and documents to guide the development of a solution for a given problem in a project. Our work will focus on projects that consist of the development of solutions to problems of existing software, which are described by issue reports.

In various project management methodologies, effort estimation depends on the experience and availability of the team members allocated for that task. The use of multiple specialists demands time and has a great impact on the cost of the project [7]. For example, in Scrum [7], an agile software development methodology, the Scrum Poker is responsible for estimating effort or relative size of goals in software development [15]. It is based on consensus among experts who participate in an estimation game. However, Scrum Poker may be time consuming due to potential disagreements between the players in the game. Usually, its participants receive use cases or user stories to derive estimates for a given software functionality [15]. Team members attempt to understand the software development goals, as well as the mechanisms required to complete such objectives. They should be able to determine accurate estimates of the effort associated with each goal.

Several ML approaches have been proposed to automatically predict the effort for the development of a given software

978-1-5090-6014-6/18/$31.00 ©2018 IEEE

functionality [16], [17], [18], [19]. However, these techniques rely on software metrics, such as cost indicators and number of lines of code, as attributes to predict the size of a new project. These product development metrics may not be promptly available on early stages of agile software development [20]. Textual descriptions of software design (e.g. user stories and issue reports) may be available before these software metrics [7]. In this context, text classifiers can be used to predict effort in agile software development with the text data available [14].

Text classification approaches are commonly based on ML techniques [21]. Such approaches generally follow four steps: document preprocessing, feature selection, feature extraction and text classification [21]. Document preprocessing includes stopword removal, stemming and pruning. Feature selection consists of finding informative subsets from the original feature set according to the importance of each attribute. Our work will focus on the feature extraction step. This step is commonly performed by the calculation of term statistics, such as bag-of-words, which is a term frequency approach that consists of vectors of word counts for each document [1]. Term-Frequency Inverse Document-Frequency (TF-IDF) and n-grams statistics are also employed to transform terms into vectors. The final step involves ML classifiers that can be used to predict text classes.

Since feature extraction is one of the most important steps in text classification [1], various methods have been proposed to tackle feature extraction in text data with ML techniques [1]. An autoencoder is a nonlinear adaptive feature extraction technique that consists of a neural network trained with an unsupervised algorithm [23], [24]. There are effective variations of autoencoders in literature that improved the performance of such a learner in finding relevant underlying structures in data [25], [26], [24], [27]. Issue reports are usually noisy texts and each document have a large amount of unique terms, irrelevant and misleading information [6]. Therefore, high dimensionality, sparsity and large amounts of noise might be particularly present in such documents in agile software development [11]. In this context, we will study the impact of autoencoders on the extraction of relevant information from software development documents. Specifically, we propose a study on the usefulness of various autoencoders in the classification of issue reports from real-world projects under an agile software development methodology. Our contributions are i) the use of autoencoders as feature extraction method in effort estimation based on issue reports of agile software projects; ii) and an empirical assessment of the impact of various autoencoders in the classification of documents from several real-world projects with high degree of textual noise and large amounts of unique words per document.

Our paper is organized as follows. Next section presents the various approaches for text classification and effort estimation. Section III describes the steps and techniques involved in effort estimation based on text data. In Section IV, we present our empirical study, report its results and discuss the effects of the highlighted techniques in the generalization ability of a ML classifier. In Section V, we describe our conclusions.

II. Related works

Several studies automated effort estimation in software development to produce more accurate and efficient predictions. ML methods are among the most used techniques for this purpose, since they can generalize estimates in problems where there is a high degree of uncertainty [22]. The work of [16] proposed the use of classifier ensembles to predict effort in software development. The authors of [28] investigated ensemble learning for variants of adjustment procedures used in analogy-based effort estimation. Such a problem was tackled with an evolutionary morphological algorithm in [29]. The proposed approach employs the dilation-erosion perceptron with an evolutionary learning process. In [17], effort estimation in software development was studied as a multiobjective ML problem. The work of [18] studied how to automatically find the relevance of past projects developed by an institution in predicting the effort demanded by new software. The authors of [19] investigated the use of clustering techniques to improve effort estimation when data of projects from single and multiple companies is available.

Neural networks are often employed in text classification [30]. They were employed to estimate the effort demanded by projects from 32 organizations [12]. In [29], the authors proposed a multilayer dilation-erosion-linear perceptron for effort estimation. Each neuron was formed of a hybrid morphological operator and a linear function. Fuzzy systems have also been employed in effort estimation. In [13], the authors employed a combination of adaptive neuro-fuzzy inference system and satin bower bird optimization algorithm to perform software development effort estimation. The work of [10] proposed a fuzzy identification method to automatically produce fuzzy membership functions and rules. In [11], the authors introduced a fuzzy logic system that is able to tackle imprecisions in early phases of software development projects [11].

Intuitively, effort estimation in software development should be obtained as far in advance as possible to avoid possible erroneous decisions [8], [20]. The ML techniques mentioned earlier use software metrics, such as size indicators, as attributes to predict the size of a new project. In fact, [12] measured the functionality size in function points and denoted effort as development hours. These product development metrics may be available only on late stages of a project. However, textual descriptions of software design, such as user stories and issue reports (in case of the correction of a software functionality), are available in the early stages of the project. In this sense, the use of text classification algorithms for the estimation of effort for a given functionality may be an important contribution to the automation of software development. Our study could help experts in the estimation stage, such as in Scrum Poker [15], to obtain more reliable and efficient predictions for this task, while employing less team members.

The work of [31] proposed a method that augments in advance both training and test documents by adding novel compound features. In [4], the authors introduced a multilayer classification framework that employs semantic and syntactic

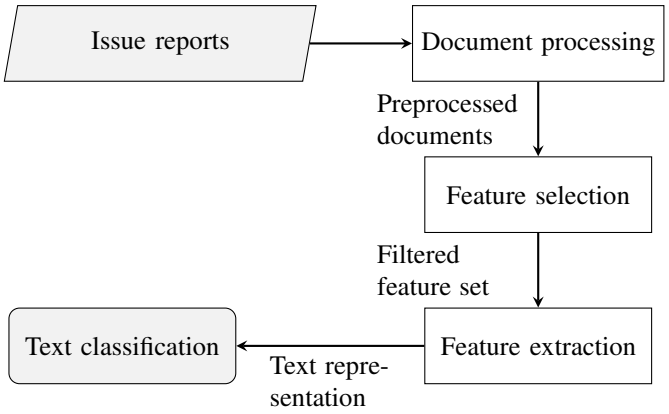

Fig. 1. Steps of effort estimation as text classification.

information represented in a two-level representation model. The work of [32] preclassified portions of texts to build classifiers that are able to minimize text temporal effects, such as class distribution, term distribution and class similarity. In [33], the authors evaluated the effectiveness of statistical keyword extraction algorithms with ensemble classifiers. Distributed Memory Paragraph Vector (PV-DM) and Distributed Bag-of-Words Paragraph Vector (PV-DBOW) are unsupervised frameworks that learn continuous distributed fixed-length vector representations for variable-length texts [34]. It outperforms bag-of-words models and other text representation techniques [34].

To the best of our knowledge, only [14] tackled effort estimation as text classification. Such an approach confirmed that a suitable classifier can be trained with issue reports in order to predict story points. However, [14] did not analyze the impact of state-of-the-art feature extraction methods. The authors employed the TF-IDF technique, which is based only on term frequencies and may not recover useful complex semantics from data [1].

Usually, the vocabulary used in issue reports can be quite extensive [20] and may include data in programming language. Consequently, the dimensionality of text classification may be very high, which may affect the performance of classification methods [23]. In order to reduce noise, dimensionality, ambiguities, redundancies and irrelevant data inherent to natural language texts, our work will investigate several autoencoder techniques [23], [25], [26], [24], [27]. Such algorithms can alleviate the impact of such problems on the learning of classification methods for effort estimation.

III. Effort estimation from issue reports

We aim at performing effort estimation as text classification using autoencoders as feature extraction methods. We follow the text classification steps shown in Figure 1. In the following sections, we describe our approach to each of these processes.

A. Document processing

In order to use texts in natural language in ML algorithms, we preprocessed the documents. We removed stopwords, such as articles, prepositions and punctuation, since they do not define contexts and result in poor generalization [1]. Accents were also ignored. We also removed obvious noise in corpus. Noisy texts have unimportant terms, such as punctuation marks, numerical values, links and URLs. We eliminated these terms from the texts to improve generalization, as the size of the feature space would be reduced. Snippets of code were left in the texts, since they may help to describe issues in software.

The selected issue reports have large amounts of unique terms, that is, many terms have very low frequencies. This occurs due to the very nature of each report, that may use issue-specific terms, often code and keywords. Therefore, we ignored terms with frequency below a threshold, which was optimized by model selection with cross-validation.

A term can be formed of more than one word or a window of characters. The n-grams are combinations of n words or characters that may provide better significance as features than single terms [1]. The parameter n was optimized by model selection with cross-validation.

We generated a dataset with term counts for each software development project. Each dataset is formed of n issue reports, denoted as instances, and d word counts (features), representing the terms in the corpus. The datasets were scaled in $[0, 1]$ to avoid features in different magnitudes [22].

B. Feature selection

In order to filter terms that do not help to distinguish classes for a text, we removed features with importance below a percentile, which was optimized by model selection. The feature importance was measured with three supervised univariate statistics, as suggested in [1] and [35]: ANOVA F-value, χ^2 statistic and mutual information.

The F-value is the test statistic in F-tests. It measures the variability between group means and within group variability [35]. The χ^2 statistic tests the independence of two events. In our study, it measures the independence of story points, denoted by classes, and terms. Such statistic is defined for term t and class c as follows [1].

$$\chi^2(t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} \frac{(N_{e_t,e_c} - E_{e_t,e_c})^2}{E_{e_t,e_c}},$$

where $e_t = 1$ if the document contains term t and 0 otherwise. If the document is in class c, then $e_c = 1$ and $e_c = 0$ otherwise. The values $N_{t,c}$ and $E_{t,c}$ indicate the observed and expected frequency for each state of term t and class c, respectively. The χ^2 statistic measures the degree in which expected counts $E_{t,c}$ and observed counts $N_{t,c}$ deviate from each other. Mutual information measures the amount of information that a term contributes to the correct classification on c [1]. Such statistic is as follows.

$$MI(t,c) = \sum_{e_t \in \{0,1\}} \sum_{e_c \in \{0,1\}} p(e_t, e_c) \log_2 \frac{p(e_t, e_c)}{p(e_t)p(e_c)},$$

where $p(.)$ is the probability function.

C. Feature extraction

Feature extraction methods attempt to find relevant underlying features in data. TF-IDF is a text statistic that attempts to reflect how important a word is in a corpus of texts. For a term t and document d, TF-IDF is defined as

$$tfidf(d,t) = tf(t) * idf(d,t),$$

where $tf(t)$ is the count of term t in document d. IDF is defined as

$$idf(d,t) = \log\left(\frac{n}{df(d,t)}\right) + 1,$$

where $df(d,t)$ is the number of documents d that contain term t and n is the number of documents. High TF-IDF denotes that a term occurs several times within a small number of documents, which generates higher discriminative relevance to this term. Low TF-IDF indicates that a term occurs few times in a document or it is present in many texts, which denotes a less informative term. TF-IDF has lowest values when a term occurs in all documents, which renders this term irrelevant.

PV-DM and PV-DBOW are neural networks able to transform high-dimensional bag-of-words representations of texts into low-dimensional vectors with an unsupervised training. Since PV-DM was effective in various domains [34], we will employ this approach in our study. In PV-DM, concepts are represented by a combination of non-mutually exclusive terms, which leads to the learning of larger amounts of concepts and the capturing of more informative underlying structures in text data. The training algorithm assumes that word vectors contribute in the predict ion of the next word in a sentence. Word vectors encode text semantics as a by-product of this prediction. Each document is denoted as an unique column vector in a matrix of documents D and each term is represented by a unique column vector in matrix of terms W. The paragraph and term vectors are combined to predict the next term in the context of each text. The document vector represents a memory that help to indicate missing parts from the current context. The length of a context is fixed and its words are sampled from a sliding window over the paragraph. PV-DM employs Stochastic Gradient Descent (SGD) via backpropagation. At each epoch, PV-DM samples a fixed-length context from a random paragraph, calculates error gradient and uses it to update the weights of the network.

An autoencoder is a nonlinear adaptive feature extraction technique that uses an encoding neural network (encoder) to transform high-dimensional input $\mathbf{x}$ into a low-dimensional code $\mathbf{h}$ and an analogous decoding neural network (decoder) to recover the data $\hat{\mathbf{x}}$ from the code [23]. The training of both encoder and decoder is unsupervised. Autoencoders can also be interpreted as nonlinear adaptive multilayer generalizations of Principal Components Analysis [23]. They can find informative representations of training data, which correspond to the transformed data $\mathbf{h}$ produced by the encoder [23]. Autoencoders with multiple layers often captures relevant hierarchical groupings or part-whole decompositions of the input, in which each hidden layer learns increasing orders of input abstractions (features) [27]. This mechanism may help the autoencoder to find proper semantics in issue reports. There are effective variations of autoencoders that find relevant underlying structures in data [25], [26], [24], [27]. In this work, several autoencoders will be employed in effort estimation from issue reports. For example, we depict the architecture of a Stacked AutoEncoder (SAE) in Figure 2.

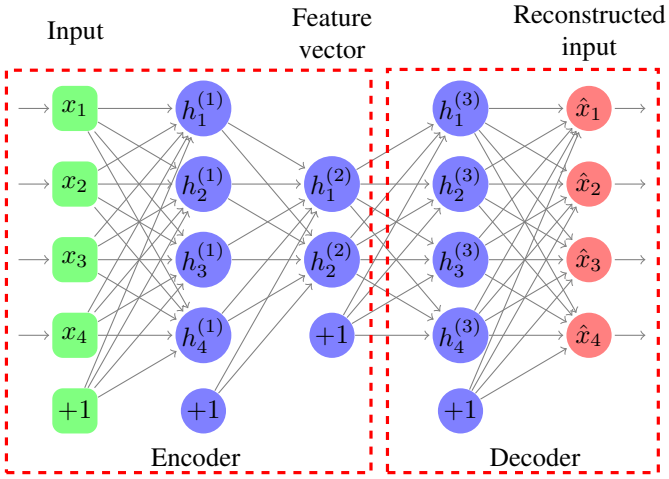

Fig. 2. Architecture of a Stacked Autoencoder.

The training of autoencoders reconstructs the input data $\mathbf{x}$, that is, it attempts to find a function $\mathbf{h}(\mathbf{W}, \mathbf{x}) = \hat{\mathbf{x}} \approx \mathbf{x}$. Its target values y_i coincide with the inputs x_i of a given instance, $i = 1, \ldots, d$, where $\mathbf{x} \in \mathbb{R}^d$. The feature vectors extracted from the input correspond to the activations $\mathbf{h}^{(2)}$ of the second hidden layer in Figure 2. The decoder network receives the representation $\mathbf{h}$ as its input and reconstructs the output $\hat{\mathbf{x}}$ as highlighted in Figure 2. Since issue reports have a large vocabulary, we aim at reducing the data dimensionality by using undercomplete autoencoders, in which the encoder produces a lower dimensional reconstruction of the input [27]. Undercomplete autoencoders impose a constraint to the training so that the activation functions $\mathbf{h}^{(i)}$ of the last layer of the encoder are a compressed representation of the input.

Autoencoders can be stacked in order to learn more complex features in different degrees of abstraction [27]. Stacked AutoEncoders (SAE) use a greedy layerwise training to update the weights $\mathbf{W}^{(i)}$ of layer i, where each layer is trained in separate stages as follows (the layerwise training can be performed, for example, by SGD).

1) The first layer is trained with raw input $\mathbf{x}$ as its input and desired output to adjust the weights $\mathbf{W}^{(1)}$ and produce the activations $\mathbf{h}^{(1)} = \hat{\mathbf{x}}$ that reconstruct $\mathbf{x}$.
2) Each subsequent layer i is trained with $\mathbf{h}^{(i-1)}$ as its input and target in order to train $\mathbf{W}^{(i)}$ and encode $\mathbf{h}^{(i)} = \hat{\mathbf{h}}^{(i-1)}$.

Such an approach trains each layer individually without considering the other layers. Each subsequent layer is trained to find increasingly complex representations of the input instances, based on the representation found by the previous

layer. Since each hidden layer adds more operations to the representation, the autoencoder gradually increases its level of abstraction. After layerwise training, we performed fine-tuning using backpropagation with SGD for all layers to improve the network generalization [24].

The performance of autoencoders can be improved by learning representations that are robust to partially corrupted input [24]. Denoising AutoEncoders (DAE) assume that more robust and meaningful features can arise from noisy instances than features extracted from the original input. We employ DAE in effort estimation as the true semantics (potential hidden features) that experts use to perform effort estimation may be hidden in the noisy language of issue reports. The feature vector $\mathbf{h}$ is trained according to a corrupted input with Gaussian noise $\tilde{\mathbf{x}} = \mathbf{x} + \gamma\mathcal{N}(0,1)$, where $\mathcal{N}(0,1)$ is the standard Gaussian distribution (with 0 mean and unit variance) and γ regulates the corruption level applied to the input. We trained DAE with SGD and its hyperparameters were optimized by randomized search [36] with cross-validation.

We employ Stacked DAE (SDAE) to capture, in each hidden layer, features with increasing levels of abstraction [24] in issue reports, such as words, n-grams, expressions, phrases, concepts and ideas.

Variational AutoEncoder (VAE) is a probabilistic multilayer generative model, where latent variables are trained in a backpropagation approach [25], [26]. The encoder neural network outputs a hidden representation $\mathbf{z}$ and it has weights and biases $\boldsymbol{\theta}$. The output follows the probability density function $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$. The lower-dimensional encoding space is stochastic, that is, the encoder produces parameters to $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$, which is often a Gaussian probability density function. We can sample from this distribution to generate noisy values of the representation $\mathbf{z}$. The input of the decoder is the representation $\mathbf{z}$. The decoder outputs the parameters to the probability distribution of the reconstructed data, and has weights and biases $\boldsymbol{\phi}$. It is denoted by $p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})$. The loss function of the VAE is the negative log-likelihood with a regularizer as follows.

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}) = E_{\mathbf{z}\sim q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})}[\log p_{\boldsymbol{\phi}}(\mathbf{x}|\mathbf{z})] + KL(q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})||p(\mathbf{z})),$$

where the first term is the expected negative log-likelihood of instance $\mathbf{x}$, which denotes its reconstruction loss and leads to the proper decoding of $\mathbf{z}$. The expectation is calculated with respect to the encoder distribution over the representations $\mathbf{z}$. If the decoder cannot reconstruct accurately, this term will incur an increase in loss $\mathcal{L}$. The second term is a regularizer based on the Kullback-Leibler divergence, $KL(.)$, between the encoder distribution $q_{\boldsymbol{\theta}}(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{z})$. Such term measures how accurately q can represent p, that is, how close these distributions are to each other. Often p is a Normal distribution with zero mean and unit variance.

D. Text classification

With informative features extracted from issue reports, we can use them as input for ML classifiers in order to perform the effort estimation. The effort is measured in story points, which correspond to classes. We analyzed several single and ensemble classifiers in our preliminary experiments. Since we focus on feature extraction, we opted to report the results of a single method, the Support Vector Machine (SVM) [22], which was the single classifier with best generalization ability in our preliminary analysis. SVM also delivered high generalization performance for text classification in [37]. Its hyperparameters were optimized via randomized search and cross-validation.

IV. Experiments

Agile development teams often adopt issue reports to describe tasks, and story points to denote task effort. Story point is a subjective metric used by agile teams to estimate the effort required to complete a development task [7]. The number of story points denotes the expected difficulty of a given task for the development team. Story points are obtained through a consensus within a team and do not represent unbiased estimates of the task difficulty [7]. In this section, we describe the selected corpora and the experimental analysis.

A. Datasets

The corpora of issue reports were selected from six open-source projects that keep record of story points [14]. These projects are examples of several scenarios of project domains, number of issues and development experiences [14].

Issue reports were extracted from issue tracking tools used by the selected projects. Each issue report is composed of several fields. Since many of these fields have irrelevant codes and identifiers, we use only the textual data in the summaries of issue reports, so that this extra noise does not mislead the training of autoencoders.

In order to select useful issue reports from each project, we used the following criteria [14]:

1) story points must have been assigned once and never updated;
2) the issue must have been addressed and finalized;
3) the values of story points must correspond to those in Planning Poker cards [15].

The projects and the respective corpora used in our study are as follows.

- (APSTUD) Aptana Studio[1], a web development Integrated Development Environment (IDE). Corpus has 203 texts and 6 classes;
- (DNN) DNN Platform[2], a web content management system. Corpus has 496 texts and 4 classes;
- (MESOS) Apache Mesos[3], a cluster manager. Corpus has 372 texts and 5 classes;
- (NEXUS) Sonatype's Nexus[4], a repository manager for software artifacts required for development. Corpus has 846 texts and 5 classes;
- (SPRINGXD) Spring XD[5], a unified distributed extensible system for data ingestion, real time analytics, batch

[1]http://www.aptana.com/
[2]http://www.dnnsoftware.com/products
[3]http://mesos.apache.org/
[4]http://www.sonatype.org/nexus/
[5]http://projects.spring.io/spring-xd/

processing, and data export. Corpus with 374 texts and 12 classes;
- (TISTUD) Appcelerator Studio[6], an IDE for mobile software. Corpus with 1148 texts and 7 classes.

B. Model selection

We used randomized search [36] to optimize the hyperparameters of all methods involved in the steps described in Section III. Each combination of hyperparameters was evaluated with 10-fold cross-validation.

In the feature selection step, we optimized the parameter n for n-grams with a discrete uniform distribution in $[1, 4]$. We considered both words and windows of characters as n-grams [5]. The minimum number of occurrences of a term (n-gram) was denoted as a fraction of documents that possess that term. The n-grams with less occurrences than that threshold were ignored. This threshold was optimized with a uniform distribution in $[1\%, 50\%]$. In the univariate feature selection, we filtered terms with less importance than a percentile. Such a percentile was optimized within a uniform distribution in $[5\%, 95\%]$. The importance score was produced by χ^2 statistic, F-score or mutual information. These functions were randomly selected with a uniform distribution.

For tuning PV-DM, we followed the suggestions of [34]. For the autoencoders and ML classifier (SVM), the randomized search followed Table I.

TABLE I
PROBABILITY DISTRIBUTIONS FOR RANDOMIZED SEARCH OF HYPERPARAMETERS.

Probability distributions for autoencoders	
Dimension of feature vectors	uniform in $[1, 100]$
Number of hidden layers	Fixed at 3
Activation functions	discrete uniform in {softmax, softplus, softsign, rectifier linear unit, hyperbolic tangent, sigmoid, hard sigmoid, linear}
Epochs	discrete uniform in $[1000, 3000]$
Epochs for SDAE	discrete uniform in $[500, 1000]$
Batch size	discrete uniform in $[30, 100]$
Learning rate	uniform in $[0, 1]$
Noise level for denoising autoencoders	uniform in $[0, 1]$
Probability distributions for SVM	
C	exponential with $\lambda = 10^{-3}$
Kernel function	discrete uniform in {linear, polynomial, sigmoid, RBF}
γ	uniform in $[0, 1]$
polynomial degree	discrete uniform in $\{1, \ldots, 4\}$

C. Results

In order to evaluate the text classification methods, we used precision, recall and F-measure, as the story points (classes) of issue reports are imbalanced. The mean and standard deviation of these rates for each feature extraction algorithm are presented in Table II.

The work of [14] employed TF-IDF to perform issue report classification. In order to assess the impact of autoencoders on effort estimation via text classification, we use TF-IDF as a benchmark [14] as it represents simple and superficial relationships between terms, which are denoted as proportions of these terms in texts. We performed the Bonferroni-Dunn post-hoc test for precision, recall and F-measure [38] with 5% of significance to compare feature extraction techniques over multiple datasets. There was no significant difference of precision between TF-IDF, PV-DM and autoencoders. All methods provided similarly relevant features to distinguish between a particular class and all others. This might indicate that the summaries of issue reports have sufficiently straightforward semantics that a simple frequency-based technique (TF-IDF) can help to elucidate and classes have relatively little overlap. All algorithms also delivered statistically similar recall rates. This might denote that the classes of efforts were properly defined with the extracted features and identified by the SVM.

Each method delivered different trade-offs between precision and recall for each dataset. In order to compare these methods with one coherent metric, we also used Bonferroni-Dunn test with F-measure, which encapsulates the trade-off between precision and recall with equal importance. SAE was statistically superior to TF-IDF. Such a result indicates that n-gram frequencies provide less informative features to the classifier when compared to SAE. Such frequencies do not consider the rules that regulate proper sequences of terms, thus TF-IDF was not able to capture most complex structures embedded in these texts, such as grammar rules and semantics. In contrast, SAE could produce more meaningful representations with higher level of abstraction for the SVM.

Summaries of issue reports are written in a very objective and succinct manner, as demonstrated in Table III. This fact alleviates some natural language challenges, such as ambiguities or symbolism. In fact, there was no significant difference between SAE, DAE, SDAE and VAE according to the Bonferroni-Dunn test. Such a result may be a consequence of the relative semantic simplicity of issue report summaries. In this context, it is expected that the ability of all employed autoencoders in revealing meaningful features would be sufficient for these texts. The denoising autoencoders, DAE and SDAE, did not help to produce superior generalization when compared to SAE and VAE. This might denote that the presence of code keywords, grammar errors and other symbols constitutes a high degree of textual noise and the addition of artificially corrupted inputs is not useful for autoencoders.

PV-DM was statistically similar to the autoencoders. It was also able to identify relevant feature vectors and help the SVM in delivering good generalization performance. VAE can be interpreted as a generative model. This characteristic can be helpful in the assessment of the features learned by the model since artificial sentences can be produced and evaluated.

Autoencoders were able to learn relevant semantics hidden with noise, code keywords and punctuation in issue reports. In Figure 3, we plot a bidimensional projection of the features learned by SAE in the DNN corpus. Each point corresponds to an issue report. We highlight three points (A, B and C) in Figure 3, these points correspond to the texts shown in Table

[6]http://www.appcelerator.com/mobile-app-development-products/

TABLE II
MEAN AND STANDARD DEVIATION OF F-MEASURE, PRECISION AND RECALL RATES OF EACH DATASET AND FEATURE EXTRACTION ALGORITHM.

Datasets	TF-IDF	PV-DM	SAE	DAE	SDAE	VAE
			Precision			
APSTUD	0.81±0.11	0.75±0.07	0.79±0.10	0.77±0.09	0.76±0.09	0.76±0.09
DNN	0.87±0.03	0.87±0.03	0.88±0.03	0.87±0.03	0.87±0.03	0.87±0.03
MESOS	0.80±0.07	0.76±0.08	0.77±0.05	0.75±0.07	0.71±0.05	0.71±0.05
NEXUS	0.85±0.01	0.85±0.02	0.85±0.01	0.85±0.01	0.85±0.01	0.85±0.01
SPRINGXD	0.76±0.04	0.73±0.06	0.69±0.06	0.70±0.06	0.59±0.12	0.73±0.05
TISTUD	0.82±0.04	0.81±0.04	0.85±0.04	0.80±0.03	0.72±0.03	0.78±0.03
			Recall			
APSTUD	0.89±0.06	0.82±0.07	0.88±0.06	0.87±0.05	0.87±0.05	0.87±0.05
DNN	0.93±0.01	0.91±0.01	0.93±0.01	0.93±0.01	0.93±0.01	0.93±0.02
MESOS	0.87±0.04	0.83±0.05	0.84±0.03	0.82±0.03	0.83±0.03	0.83±0.03
NEXUS	0.92±0.01	0.92±0.01	0.92±0.01	0.92±0.01	0.92±0.01	0.92±0.01
SPRINGXD	0.80±0.05	0.75±0.05	0.75±0.05	0.76±0.06	0.66±0.03	0.76±0.05
TISTUD	0.87±0.01	0.85±0.02	0.88±0.02	0.83±0.02	0.85±0.01	0.85±0.02
			F-measure			
APSTUD	0.81±0.07	0.78±0.07	0.83±0.08	0.82±0.07	0.81±0.07	0.81±0.07
DNN	0.90±0.02	0.91±0.03	0.91±0.02	0.90±0.02	0.90±0.02	0.90±0.02
MESOS	0.76±0.05	0.79±0.06	0.80±0.04	0.80±0.05	0.76±0.04	0.77±0.04
NEXUS	0.88±0.01	0.88±0.01	0.88±0.01	0.88±0.01	0.88±0.01	0.88±0.01
SPRINGXD	0.71±0.05	0.73±0.06	0.71±0.06	0.72±0.06	0.70±0.05	0.73±0.06
TISTUD	0.84±0.03	0.82±0.03	0.85±0.03	0.81±0.02	0.78±0.02	0.81±0.03

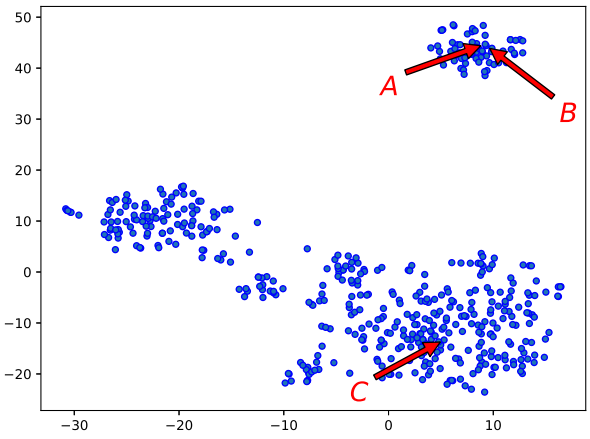

Fig. 3. Bidimensional projection of features of the DNN issue reports revealed by SAE.

TABLE III
SUMMARIES OF THE ISSUE REPORTS HIGHLIGHTED IN FIGURE 3.

A	*SI: Search Results are showing code (Javascript etc.)*
B	*SI: PB Users: Search Does Not Work When Showing All Users*
C	*'Existing' page type is showed as 'tab'*

III. The feature vectors of texts A and B are close to each other and point C is far from that pair. Texts A and B have similar semantics: both issues are related to how search results are shown. Intuitively, different texts about the same module are likely to share terms and semantics. In fact, issue reports A and B describe problems with the same software module, namely the *search module*, and text C reports a problem with the *pages module*.

Therefore, the distinction of contexts (in this case, software modules) increases the amount and quality of information provided to the classifier, so that such an algorithm can equate this potentially meaningful feature to effort categories. The learning of such a software component context is an example of how autoencoders can identify features in different levels of abstraction and improve effort estimation. Other useful features might have low level of abstraction, such as the identification of code keywords; or high level of abstraction, such as the distinction between software errors and suggestions from users. Moreover, SAE revealed three clear groups of texts in Figure 3, which might indicate other important underlying features that might be related to the effort required for each software correction and may be used for clustering purposes.

V. CONCLUSIONS

Software development effort estimation is an important task for the delivery of high-quality software products within a budget and a deadline. Such estimation is also performed for the correction of issues in software, which can be considered as part of the software development. Intuitively, the estimates should be produced as soon as possible so that the development team can successfully plan and implement the software solution. Issue reports are textual descriptions of software problems, they are the earliest documents available in the development of a correction of software. Such texts are categorized by agile teams using story points to denote the difficulty of the development of solutions for issues. In this sense, we investigated automatic software development effort estimation as text classification using autoencoders.

Autoencoders can learn relevant features in various degrees of abstraction. Such methods can be used in text classification for finding useful semantics that may be used in the training of a classifier. In this work, we studied several autoencoders in the context of effort estimation and compared these methods with TF-IDF and PV-DM. SAE was able to produce informative

features, which significantly improved the generalization performance of the SVM in comparison with TF-IDF. There was no statistical difference between autoencoders. These results demonstrated the effectiveness of autoencoders in learning features in increasing degrees of abstraction. In fact, SAE could learn, for example, code keywords (features with low level of abstraction) and the software modules to which the issue reports were associated (more complex features). Such text features provided discriminative information to the classifier, so that it could associate these learned representations to the required effort of a given issue.

In future studies, we aim to investigate the use of the description field in issue reports to aggregate potentially useful information to the effort estimation pipeline. We also intend to tackle effort estimation as text classification in the development of entirely new software projects, using development documents and deeper architectures for autoencoders. Other open-source and industrial projects may also be considered.

REFERENCES

[1] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] C. C. Aggarwal and C. Zhai. *A Survey of Text Classification Algorithms*, chapter 2, pages 163–222. Springer US, Boston, MA, 2012.

[3] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang. Systematic literature review of machine learning based software development effort estimation models. *Information and Software Technology*, 54(1):41–59, 2012.

[4] J. Yun, L. Jing, J. Yu, and H. Huang. A multi-layer text classification framework based on two-level representation model. *Expert Systems with Applications*, 39(2):2035–2046, 2012.

[5] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 1st edition, June 1999.

[6] M. Jorgensen and S. Grimstad. The impact of irrelevant and misleading information on software development effort estimates: A randomized controlled field experiment. *IEEE Transactions on Software Engineering*, 37(5):695–707, September 2011.

[7] M. Cohn. *Agile estimating and planning*. Prentice Hall, 2005.

[8] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *IEEE Transactions on Software Engineering*, 33(1):33–53, Jan 2007.

[9] R. A. Araújo, A. Oliveira, and S. Meira. A class of hybrid multilayer perceptrons for software development effort estimation problems. *Expert Systems with Applications*, 90:1–12, 2017.

[10] Z. Xu and T. M. Khoshgoftaar. Identification of fuzzy models of software cost estimation. *Fuzzy Sets and Systems*, 145(1):141–163, 2004. Computational Intelligence in Software Engineering.

[11] Z. Muzaffar and M. A. Ahmed. Software development effort prediction: A study on the factors impacting the accuracy of fuzzy logic systems. *Information and Software Technology*, 52(1):92–109, 2010.

[12] G. Wittig and G. Finnie. Estimating software development effort with connectionist models. *Information and Software Technology*, 39(7):469–476, 1997.

[13] S. Moosavi and V. Bardsiri. Satin bowerbird optimizer: A new optimization algorithm to optimize anfis for software development effort estimation. *Engineering Applications of Artificial Intelligence*, 60:1–15, 2017.

[14] S. Porru, A. Murgia, S. Demeyer, M. Marchesi, and R. Tonelli. Estimating story points from issue reports. In *Proceedings of the 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2016, pages 1–10, New York, NY, USA, 2016. ACM.

[15] J. Grenning. Planning poker or how to avoid analysis paralysis while release planning. *Hawthorn Woods: Renaissance Software Consulting*, 3, 2002.

[16] L. Minku and X. Yao. Ensembles and locality: Insight on improving software effort estimation. *Information and Software Technology*, 55(8):1512–1528, 2013.

[17] L. Minku and X. Yao. Software effort estimation as a multiobjective learning problem. *ACM Transactions on Software Engineering and Methodology*, 22(4):1–32, October 2013.

[18] L. Minku and X. Yao. Which models of the past are relevant to the present? a software effort estimation approach to exploiting useful past models. *Automated Software Engineering*, 24(3):499–542, September 2017.

[19] L. Minku and S. Hou. Clustering dycom: An online cross-company software effort estimation study. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE, pages 12–21, New York, NY, USA, 2017. ACM.

[20] R. C. Lee. The success factors of running scrum: A qualitative perspective. *Journal of Software Engineering and Applications*, 5(6), June 2012.

[21] F. P. Shah and V. Patel. A review on feature selection and feature extraction for text classification. In *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 2264–2268, March 2016.

[22] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786), 2006.

[24] P. Vincent, H. Larochelle, Y. Bengio, and P. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1096–1103, New York, NY, USA, 2008. ACM.

[25] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[26] D. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, ICML'14, pages 1278–1286. JMLR.org, 2014.

[27] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11:3371–3408, December 2010.

[28] M. Azzeh, A. Nassif, and L. Minku. An empirical evaluation of ensemble adjustment methods for analogy-based effort estimation. *Journal of Systems and Software*, 103:36–52, 2015.

[29] R. A. Araújo, A. L. I. Oliveira, S. Soares, and S. Meira. An evolutionary morphological approach for software development cost estimation. *Neural Networks*, 32:285–291, 2012.

[30] Y. Goldberg. *Neural Network Methods in Natural Language Processing (Synthesis Lectures on Human Language Technologies)*. Morgan & Claypool Publishers, April 2017.

[31] F. Figueiredo, L. Rocha, T. Couto, T. Salles, Marcos André Gonçalves, and Wagner Meira Jr. Word co-occurrence features for text classification. *Information Systems*, 36(5):843–858, 2011.

[32] L. Rocha, F. Mourão, A. Pereira, M. A. Gonçalves, and W. Meira, Jr. Exploiting temporal contexts in text classification. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 243–252, New York, NY, USA, 2008. ACM.

[33] A. Onan, S. Korukoğlu, and H. Bulut. Ensemble of keyword extraction methods and classifiers in text classification. *Expert Systems with Applications*, 57:232–247, 2016.

[34] Q. Le and T. Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32, pages 1188–1196, Beijing, China, June 2014. PMLR.

[35] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson, 6th edition, April 2007.

[36] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13, 2012.

[37] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In Claire Nédellec and Céline Rouveirol, editors, *Machine Learning: ECML-98*, pages 137–142, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[38] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, December 2006.