Extracting Features from Requirements: Achieving Accuracy and Automation with Neural Networks

Yang Li
Otto-von-Guericke University
Magdeburg, Germany
Email: yang.li@ovgu.de

Sandro Schulze
Otto-von-Guericke University
Magdeburg, Germany
Email: sanschul@ovgu.de

Gunter Saake
Otto-von-Guericke University
Magdeburg, Germany
Email: saake@ovgu.de

***Abstract*—Analyzing and extracting features and variability from different artifacts is an indispensable activity to support systematic integration of single software systems and Software Product Line (SPL). Beyond manually extracting variability, a variety of approaches, such as feature location in source code and feature extraction in requirements, has been proposed for automating the identification of features and their variation points. While requirements contain more complete variability information and provide traceability links to other artifacts, current techniques exhibit a lack of accuracy as well as a limited degree of automation. In this paper, we propose an unsupervised learning structure to overcome the abovementioned limitations. In particular, our technique consists of two steps: First, we apply Laplacian Eigenmaps, an unsupervised dimensionality reduction technique, to embed text requirements into compact binary codes. Second, requirements are transformed into a matrix representation by looking up a pre-trained word embedding. Then, the matrix is fed into CNN to learn linguistic characteristics of the requirements. Furthermore, we train CNN by matching the output of CNN with the pre-trained binary codes. Initial results show that accuracy is still limited, but that our approach allows to automate the entire process.**

I. Introduction

A *Software product line (SPL)* allows to develop a family of products that share common functionalities and characteristics satisfying the specific needs of a particular market segment [1]. To this end, Software Product Line Engineering (SPLE) describes a systematic development process taking commonalities and differences (in terms of features) between particular products into account [2], [3]. A feature in this context is a user-visible increment in functionality [4]. While SPLs have been proven to be beneficial, it is not used for initial development due to a) high upfront cost and b) missing certainty whether a large amount of variants is needed. Instead, traditional and more unstructured techniques for reuse, such as *clone-and-own* [5], [6] are used to cope with diverging products in the beginning. However, at this point, information about commonalities and variability is usually not explicitly given, but rather hidden within the artifacts. Hence, in case of a transition from unstructured to structured reuse using SPLs, this information needs to be recovered.

Reverse engineering techniques, such as feature location and feature extraction, are common means to support automatic extraction of features and, in some cases, even their variability information. While feature location has been subject to intense research [7], it exhibits crucial limitations that hinder applicability in an SPL context. In particular, 1) the majority of existing feature location techniques focus on source code in single software systems, and thus, do not recover variability information, and 2) most of the techniques are tailored to source code, and thus, not applicable for other artifacts.

Recently, approaches that focus on requirements to recover variability information have been proposed [8]–[16], because requirements 1) contain more comprehensive information about commonalities and variability, and 2) establish traceability links to other artifacts of later development phases (e.g., source code). Some of these approaches highly depend on the syntactic information of sentences (i.e., parse trees) [11]–[16], which requires manual intervention or external tools (e.g., Stanford Parser [17]), while other approaches apply external knowledge resources such as WordNet [18] to identify synonyms and compute the similarity of each pair of sentences [12]–[15]. However, all of these approaches lack either accuracy (i.e., assign requirements to wrong features) or a sufficient degree of automation, thus, requiring extensive manual intervention [19].

To overcome these limitations, we propose a technique based on a unsupervised learning structure to extract features. Basically, our technique consists of three parts, 1) a Convolutional Neural Network (CNN), 2) an unsupervised dimensionality reduction function, and 3) a Clustering algorithm. Using this technique allows to make use of a compressed binary representation of the requirements and to apply word embedding models as prediction models, which have shown to outperform common count models [20].

In this paper, we focus on the abovementioned steps 1 and 2, thus, making the following contributions:

- We propose a technique to extract features from requirements that 1) utilizes an unsupervised learning structure *without* relying on syntax information and external knowledge resources, and 2) integrates a prediction model to process raw text requirements and generate word vectors instead of using traditional distributional semantic models (DSMs).
- We provide insights in a preliminary feasibility study and discuss current results and possible improvements we are currently working on.

978-1-5386-4969-5/18 © 2018 IEEE

SANER 2018, Campobasso, Italy
ERA Track

II. Background

In the process of creating and evolving an SPL, a feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option [4]. To represent relations and dependencies between features, *feature models (FM)* have been proposed. In particular, an FM constitutes a tree-like structure, depicting all features of a domain together with their relationships, and thus, it is a center piece in the SPLE process (e.g., for deriving valid configurations).

The following types of relationships exist:

- *Mandatory* - feature is required in each config;
- *Optional* - feature is optional;
- *AND* - all subfeatures must be selected;
- *OR* - one or more subfeatures can be selected;
- *Alternative (XOR)* - exactly one subfeature must be selected.

In addition to the above relationships between features, cross-tree constraints (CTCs) express dependencies between features across the whole FM. The most common are:

- *A requires B* - The presence of feature A implies the presence of feature B;
- *A excludes B* - The presence of feature A implies the absence of feature B.

III. Methodology

In this section, we provide an overview of our approach and introduce the particular techniques and how they are applied in our context. We provide an overview of the entire process in Fig. 1. Initially, the input documents (i.e., textual requirements specifications) are pre-processed, that is, we decompose the documents into sentences and then remove stop words in these sentences. Subsequently, the pre-processed requirements are further processed in two ways: First, by applying *Laplacian Eigenmaps* algorithm to obtain a low dimensional representation, which is then converted into binary codes. And by applying a word embedding model that creates a word vector for each word in the requirements. Next, requirements are projected into a matrix representation, by employing the pre-trained word embedding set, and fed into a CNN. Finally, the output of the CNN is compared with the binary codes to evaluate the goodness of the CNN. This process is executed repeatedly and eventually, the result can be used to utilize clustering algorithms, such as k-means, to extract features based on the characteristic representation. In the context of feature extraction from requirements, we regard features as domain artifacts, and thus, we assume that several requirements, related to the same functionality, belong to a particular feature. Next, we explain each step in more detail, with the clustering being out of scope for this paper.

A. Laplacian Eigenmaps

Laplacian Eigenmaps (LE) is applied to compute a low-dimensional representation of the input dataset (i.e., requirements) that optimally preserves local neighborhood information in a certain sense [21]. The foundation of this technique is

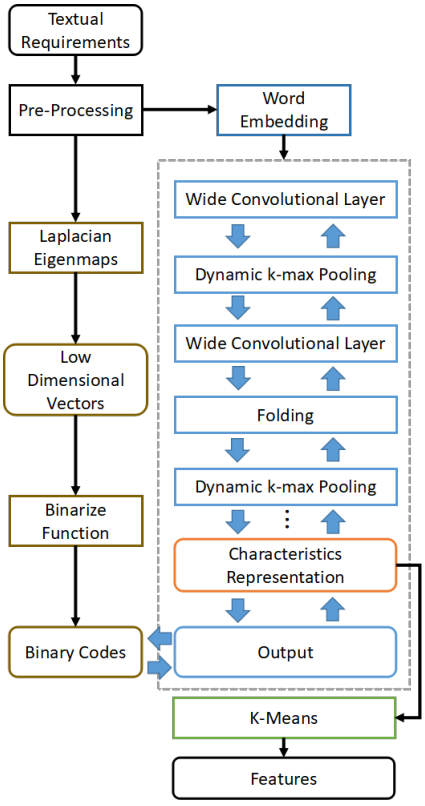

Fig. 1: The flow chart of feature extraction

the *laplacian matrix*, which resembles a graph, based on the input data. Hence, LE constitute a dimensionality reduction function based on graphs with nodes being data points and edges connecting nodes that exhibit a particular similarity (for a given similarity definition). Moreover, the edges have associated weights, indicating the degree of similarity between the connected nodes, and the similarity of any two nodes can be measured by the distance between them. Given these properties, the goal of applying LE is to reduce dimensionality by keeping similar nodes with short distance, as they are likely to exhibit a relation as well in the original data.

To achieve a graph representation, we first create a vector of each sentence by TF-IDF and then use k-Nearest Neighbors (KNN) algorithm to link each vector to its k nearest vectors. The similarity matrix A is computed by applying heat kernel algorithm, that is, A is the adjacency matrix of the KNN graph for the raw requirements. Given a diagonal $n \times n$ matrix D (i.e., $D_{ii} = \sum_{j=1}^{n} A_{ij}$), the graph Laplacian L can be calculated by $D - A$. The matrix D denotes the requirement significance: the bigger the value of D_{ii} is, the more significant is the i-th requirement which is strongly connected by neighbours. The optimal low dimensional matrix Y can be obtained by solving the following objective function:

$$\begin{aligned} \underset{Y}{arg\ min} \quad & trace(YLY^T) \\ subject\ to \quad & YDY^T = I \\ & YD1 = 0 \end{aligned} \tag{1}$$

In order to match the output of neural network in the training process, we convert the low dimensional matrix Y into a set of binary codes B [22].

B. Convolutional Neural Network

For our purposes. we utilize a specific kind of CNN, that is, a *Dynamic Convolutional Neural Network* (DCNN), which has been proposed for modeling sentences [23]. By using this model, the order of words in the sentences is preserved and word relations of varying size can be captured. In particular, the DCNN captures word relations independent of their distance in a sentence without any prior knowledge.

In Fig. 1 (right part), we illustrate how we employ the DCNN model for feature extraction from requirements. Initially, we transform our requirements into a word vector representation using the already mentioned word embedding. As a result, we obtain a set of word vectors E, each one characterizing a word in our requirements (i.e., the similarity wrt. all other words).

Next, we employ the word vectors E to transform each sentence s_i from the requirements into a matrix $S \in \mathbb{R}^{d_w \times s}$, where d_w is the dimensionality of the word vector and s is the length of a sentence. We then feed this matrix S into our DCNN, where it is processed consecutively by different layers (e.g., in Fig. 1 we show two exemplary convolutional layers). Finally, the output of one iteration of our DCNN is compared with the binary codes from our LE reduction. The result is then propagated back to adjust weight matrices inside our DCNN model.This process is repeated several times in order to learn an accurate characteristic representation of the requirements. The particular layers of our DCNN model work are as follows.

1) Wide Convolutional Layer: In our DCNN, the convolution is a matrix computation between a *weight matrix*, called *filter*, and our input matrix S. We apply a *one-dimensional* convolution to each row of matrix S, that is, we use a one-dimensional filter to scan each sentence in order to detect its latent semantic and structure information.

There are two types of convolution: *narrow* convolution and *wide* convolution. For instance, given a sentence with seven words and a filter of length five, we can execute the narrow convolution operation only three times, because the filter must be *entirely* inside the sentence. In contrast, for wide convolution, the filter is allowed to be partially outside the sentence with all elements out of range being zero (i.e., zero padding). As a result, we can execute the wide convolution operation eleven times. Consequently, wide convolution is capable of gaining more information at the margins of the sentence, because it make sure that each weight in the filter scans all the words in the sentence.

More formally, in our wide convolutional layer(s), a filter $\mathbf{f} \in \mathbb{R}^f$ slides over each row of the requirement matrix $S \in \mathbb{R}^{d_w \times s}$, eventually resulting into a matrix $C \in \mathbb{R}^{d_w \times (s+f-1)}$, where f is the width of the filter and s is the length of the sentence. The matrix C is called the *characteristic map* of the requirement matrix S.

2) Folding: So far, the filter is only applied to each row of the requirement matrix S independently, thus, neglecting the relation between different rows. To overcome this drawback, *Folding* is a method to detect latent relationships between adjacent rows: At each folding layer, every two rows in a characteristic map are summarized columnwise. Consequently, for a characteristic map with d_w rows, folding returns a map $\hat{C} \in \mathbb{R}^{(d_w/2) \times (s+f-1)}$ with only $d_w/2$ rows.

3) Dynamic k-max pooling: K-max pooling is a downsampling strategy in CNN to reduce the dimensionality of the intermediate layer output matrix (i.e., our characteristic map). This technique helps to avoid over-fitting by providing an abstracted form of our charateristic map. As a consequence, it reduces the computational costs since it reduces the parameters that are subject to computation (i.e., the entries of the map). The traditional k-max pooling operation is to pool the k most active characteristics (i.e., the most important characteristics) in a given sequence. Given a preset k, the submatrix $\bar{C} \in \mathbb{R}^{(d_w/2) \times k}$ of the k highest values in each row of the matrix $\hat{C}$ are chosen by k-max pooling. However, we apply *dynamic* k-max pooling, where the pooling parameter k is dynamically selected. As a result, we achieve a smooth extraction of higher-order and longer-range characteristics. Given a fixed, predefined pooling parameter k_{top} for the topmost convolutional layer, the parameter k of k-max pooling in the l-th convolutional layer can be computed as a function:

$$k_l = max(k_{top}, [\frac{L-l}{L} s]) \tag{2}$$

where L is the total number of convolutional layers in the network.

4) Output: The output layer is a key step to complete the learning process by fitting the binary codes B, and thus,realizing the unsupervised learning process. The output layer of DCNN is a function:

$$O = W_O h \tag{3}$$

with (i) h being the characteristic representation, (ii) $O \in \mathbb{R}^q$ being the output vector, and (iii) $W_O \in \mathbb{R}^{q \times r}$ being the weight matrix. Finally, we train the DCNN model by back-propagation and speed up training by using Adam optimization algorithm [24].

C. K-Means

Given the linguistic characteristic representation of the requirements, learned by the DCNN, the result can be used with clustering algorithms, such as traditional K-means algorithm, to group sentences which describe similar functionality into a cluster based on the characteristic representation. In particular, our intuition is that the requirements, represented by a particular cluster together, belong to the same feature.

IV. Related Work

Dependency on syntax and external knowledge database: Reinhartz-Berger et al. proposed an approach for analyzing features and variability by combining semantic similarity with

similarity of software behavior as manifested in requirement statements [11]. To this end, they applied Semantic Role Labeling (SRL) techniques that highly relies on syntactic information. Itzik et al. also used SRL to transform each sentence into six roles and computed similarity of each pair of semantic roles by applying WordNet [12]. Afterwards, Hierarchical Agglomerative Clustering (HAC) is applied to these roles to cluster features. Based on the papers above [11], [12], Itzik et al. proposed an approach named semantic and ontological variability analysis (SOVA) to analyze variability of functional requirements [15]. This approach uses ontological and semantic considerations to automatically analyze differences between initial states, external events, and final states of behaviors, and thus, identify features [15]. Wang proposed a method to build semantic frames for frequent verbs appearing in requirements by applying SRL with the assistance of Stanford Parser and WordNet [13], [14]. However, Wang's research only extracted semantic information of requirements and didn't extract features in the context of SPLs.

All of these approaches require syntactic information and external knowledge resources to obtain accurate semantic information of requirements. In contrast, we utilize an unsupervised learning structure (i.e., a CNN) to gain linguistic characteristic representation *without* assistance of syntactic information and external knowledge database.

Traditional DSMs: Alves et al. conducted an exploratory study on leveraging information retrieval techniques for feature and variability extraction [8]. They presented a framework by employing LSA and VSM technique to measure the similarity between sentences and also applied HAC to cluster features based on this similarity. Weston et al. proposed a tool suite for processing requirements into candidate feature models, which can be refined by domain engineers [9]. They applied LSA to measure similarity and HAC to cluster similar texts into same feature groups to create a feature tree. Moreover, they built a variability lexicon and grammatical patterns to detect latent variability information. Tsuchiya et al. proposed an approach to recommend traceability links between sentences in requirements and design-level UML class diagrams as structure models [10]. They used VSM to determine similarity between sentences in requirements. However, their research direction is not to discover features based on the similarity.

In contrast to the research above, we utilize word embedding instead of using traditional DSMs to gain word vector representation of the requirements. Also, we apply CNN to learn the linguistic characteristic representation rather than directly computing the similarity of each pair of sentences.

V. Preliminary Result and Discussion

To demonstrate the general feasibility of our approach, we implemented a prototype and applied it to a small set of requirements. In this section, we briefly state on the dataset used, the setting for our study, elaborate on initial results, and discuss them regarding accuracy and automation.

Dataset: We obtained the requirements from the *Body Comfort System (BCS)* [25], a case study from a real-world scenario that takes variability into account. Overall, BCS comprises 95 requirements with 117 sentences, specifying the functionality of the system. Moreover, BCS comes with a feature model encompassing 27 features manually created by domain engineers. Hence, in future work we can make use of this model as a ground truth to further evaluate the accuracy of our approach (e.g., when taking clustering into account).

Experiment setting and evaluation metrics: We use an API from gensim [26] that implements the *word2vec* technique, to implement word embedding, and keep all default parameters. In order to obtain pre-trained word vectors of high quality, we apply Wikipedia dumps (https://dumps.wikimedia.org/), an open source corpus with three billion words, to train word embedding. We apply cross-entropy loss function to evaluate the training process to gain the expected DCNN model [27].

Preliminary result: Up to now, we don't use K-means to cluster features in terms of the characteristic representation from current DCNN model. The key problem is that the output of DCNN doesn't match the binary code very accurately. The loss value, which is used to evaluate DCNN model's performance, reamins around 0.5 instead of continuously decreasing after approximately 1500 training steps. Clustering features based on a bad performance model makes no sense for the accuracy of extracted features.

Discussion: We discuss the results from above with respect to our goal, i.e., to achieve high accuracy and full automation.

1) For the accuracy part, we obtain a DCNN model with high loss, which means that we initially fail to achieve this goal. However, even if this is not the desired result, it was somewhat expected at this stage of our research. Basically, we identified two reasons for the rather low accuracy. First of all, our DCNN model has several parameters in the particular layers, such as weights or size of the convolution filter, that need to be tuned. While there are numerous other approaches that elaborate on these parameters, none of these approaches focus on requirements or feature location therein. Hence, these parameters need to be determined experimentally and based on a sufficiently large data set. Consequently, our initial parameters serve only as baseline (i.e., lower bound), and thus, need to be refined by applying it to more requirements and gain insights on characteristics that may effect the parameter setting. Another reason, leading to the high loss, is that relatively small size of the dataset we used. Usually, CNNs require a sufficiently large dataset for the training phase to achieve reasonable results. With only 117 sentences, we have clearly a too small sample for training a model that achieves accurate results. We are currently seeking more requirements to extend our dataset, and thus, overcome this limitation that affects the accuracy. Nevertheless, we argue that even with the current accuracy we perform better than some other approaches with the same objective, as stated by Li et al. [19].

2) For the automation part, we propose an unsupervised learning structure without syntactic information of sentences, because this often impose manual analysis and correction. Neglecting the setup of parameters, we argue that the entire learning process is automatically conducted by DCNN model.

Although we currently fail to obtain a sufficient accuracy by our DCNN model, and thus, do not apply clustering, the process of feature clustering by K-Means can be fully automating as well. Nevertheless, some manual semantic analyses may still be needed to gain the final features in some cases. For instance, this may be necessary when the sentences in a cluster belong to non-functional requirements or even are not related to a certain software (e.g., sentences w.r.t. stakeholders).

Although we may not achieve full automation in all cases, we argue that our proposed approach is capable of minimizing manual intervention, and thus, highly auomates the process of extracting features from requirements.

VI. Summary and Future Work

In this paper, we proposed an unsupervised learning structure to extract features from requirements. To this end, we combine CNN and LE to detect the linguistic information of requirements. Then, features can be extracted from the learned linguistic information by applying K-Means clustering algorithm. In particular, we aim at improving current limitations regarding accuracy and automation without any dependency on syntactic information and external knowledge database. To this end, we apply a prediction model to build word vectors, which outperforms traditional distributional semantic models (DSMs), and introduce a CNN technique for modeling sentences and to learn their linguistic characteristics. While we are struggling with the accuracy of our approach, we achieve a mostly automated process, thus, minimizing manual intervention for extracting features from requirements.

In future work, we intend to complete the current research by improving the methodology, designing a reasonable case study and evaluating the results. So far, our work focuses on feature extraction from requirements, but the variability information, such as relationship between features, is not detected. Hence, based on the features from requirements, we plan to apply a certain combination between association rule mining and reinforcement learning algorithms to extract variability information. Moreover, we intend to design a reasonable evaluation metric to measure the accuracy of extracted features and variability. Considering manually extracting feature and variability, this may pose the risk of uncertainty and bias due to differences between the results from two (or more) domain engineers. Hence, a convincing evaluation metric plays a key role in estimating if the results are accurate or not.

References

[1] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *Proc. Int'l Conf. Advanced Information Systems Engineering*. Springer-Verlag, 2005, pp. 491–503.

[2] P. C. Clements and L. M. Northrop, *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 2001.

[3] K. Pohl, G. Böckle, and F. van Der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[4] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-Oriented Software Product Lines*. Springer, 2013.

[5] Y. Dubinsky, J. Rubin, T. Berger, S. Duszynski, M. Becker, and K. Czarnecki, "An Exploratory Study of Cloning in Industrial Software Product Lines," in *Proc. Eur. Conf. Soft. Maint. and Reeng.* IEEE, 2013, pp. 25–34.

[6] C. Riva and C. D. Rosso, "Experiences with software product family evolution," in *Proc. Int'l Workshop on Principles of Software Evolution*. IEEE, 2003, pp. 161–169.

[7] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[8] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler, "An exploratory study of information retrieval techniques in domain analysis," in *Proc. Int'l Software Product Line Conference*. IEEE, 2008, pp. 67–76.

[9] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," *Proc. Int'l Software Product Line Conference*, pp. 211–220, 2009.

[10] K. Kumaki, R. Tsuchiya, H. Washizaki, and Y. Fukazawa, "Supporting commonality and variability analysis of requirements and structural models," in *Proc. Int'l Software Product Line Conference*. ACM, 2012, pp. 115–118.

[11] I. Reinhartz-Berger, N. Itzik, and Y. Wand, "Analyzing variability of software product lines using semantic and ontological considerations," in *Proc. Int'l Conf. Advanced Information Systems Engineering*. Springer, 2014, pp. 150–164.

[12] N. Itzik and I. Reinhartz-Berger, "Generating feature models from requirements: Structural vs. functional perspectives," in *Proc. Int'l Software Product Line Conference*. ACM, 2014, pp. 44–51.

[13] Y. Wang, "Semantic information extraction for software requirements using semantic role labeling," in *Proc. Int'l Conf. Progress in Informatics and Computing (PIC)*. IEEE, 2015, pp. 332–337.

[14] ——, "Automatic semantic analysis of software requirements through machine learning and ontology approach," *J. Shanghai Jiaotong University*, vol. 21, pp. 692–701, 2016.

[15] N. Itzik, I. Reinhartz-Berger, and Y. Wand, "Variability analysis of requirements: Considering behavioral differences and reflecting stakeholders' perspectives," *IEEE Trans. Soft. Eng.*, vol. 42, pp. 687–706, 2016.

[16] M. Hamza and R. J. Walker, "Recommending features and feature relationships from requirements documents for software product lines," in *Proc. Int'l Workshop Realizing Artificial Intelligence Synergies in Soft. Eng.* IEEE, 2015, pp. 25–31.

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proc. Conf. Association for Computational Linguistics (ACL)*, 2014, pp. 55–60.

[18] G. A. Miller, "Wordnet: A lexical database for english," *Comm. ACM*, vol. 38, pp. 39–41, Nov 1995.

[19] Y. Li, S. Schulze, and G. Saake, "Reverse engineering variability from natural language documents: A systematic literature review," in *Proc. Int'l Conf. Systems and Software Product Line Conference*. ACM, 2017, pp. 133–142.

[20] M. Baroni, G. Dinu, and G. Kruszewski, "Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors," in *Proc. Conf. Association for Computational Linguistics (ACL)*. ACL, 2014, pp. 238–247.

[21] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, pp. 1373–1396, Jun 2003.

[22] D. Zhang, J. Wang, D. Cai, and J. Lu, "Self-taught hashing for fast similarity search," in *Proc. ACM SIGIR Int'l Conf. Research and Development in Information Retrieval*. ACM, 2010, pp. 18–25.

[23] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," 2014.

[24] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv e-prints*, Dec 2014.

[25] S. Lity, R. Lachmann, M. Lochau, and I. Schaefer, "Delta-oriented software product line test models - the body comfort system case study," TU Braunschweig, Tech. Rep., 2013.

[26] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proc. Int'l LREC Workshop on New Challenges for NLP Frameworks*. ELRA, 2010, pp. 45–50.

[27] P. Golik, P. Doetsch, and H. Ney, "Cross-entropy vs. squared error training: a theoretical and experimental comparison," in *Proc. Int'l Conf. Speech Communication Association*. ISCA, 2013.