Software Defect Prediction Model Based on Stacked Denoising Auto-Encoder

Yu Zhu[1(✉)], Dongjin Yin[2], Yingtao Gan[2], Lanlan Rui[1], and Guoxin Xia[1]

[1] State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China
915808818@qq.com, 1912696398@qq.com, 736070995@qq.com
[2] Beijing TangMIX Technology CO., Ltd., Beijing, China
576678349@qq.com, 541629067@qq.com

Abstract. Software defect prediction technology plays an important role in ensuring software quality. The traditional software defect prediction model can only perform "shallow learning" and cannot perform deep mining of data features. Aiming at this problem, we use the stacked denoising auto-encoder (SDAE) to superimpose into deep neural network. First, the deep network model was built through the stacked layers of denoising auto-encoder (DAE), then the unsupervised method was used to train each layer in turn with noised input for more robust expression, characteristics were learnt supervised by back propagation (BP) neural network and the whole net was optimized by using error back propagation. Simulation experiments prove that the prediction accuracy of our SDAE model is significantly improved compared with the traditional SVM and KNN prediction model.

Keywords: Software defect prediction · Stacked denoising auto-encoder · Deep learning

1 Introduction

With the wide application of computer software, the quality and reliability of software are increasingly valued. According to a report [1] by the National Institute of Standards and Technology, the success rate of complex specialized application software development in the United States is only 30%, and software defects cause the US economy to lose $55.9 billion annually. Therefore, how to establish a reasonable software defect prediction model is the focus of our research.

In recent years, many researchers have conducted various researches on software defect prediction technology, and proposed software defect prediction models based on machine learning and statistics. In 1993, Briand et al. [2] applied logistic regression, classification tree and OSR methods to the defect prediction study on 146 components of the ADA system. In 1998, Evett et al. [3] first applied genetic methods to the prediction of defects in military communication systems and telecommunication systems. The Multi-Layer Perception (MLP) proposed by Pizzi et al. [4] in 2002 is an effective software defect technique. Mahaweerawa et al. [5] first used fuzzy clustering

S. Han et al. (Eds.): AICON 2019, LNICST 287, pp. 18–27, 2019.
https://doi.org/10.1007/978-3-030-22971-9_2

to predict software defects in 2002. He applied Radial Basis Function (RBF) to predict software defects. In 2014, Jindal et al. [6] established a neural network prediction model to study software defects. Various optimization models for neural networks were proposed, such as PSO-BP and SA-BP. Yang et al. [7] introduced deep learning into the software defect prediction technology in 2015. In 2017, AV Phan et al. [8] proposes to automatically learn defect features for software defect prediction using precise graphs representing program execution flows, and deep neural networks.

Inspired by previous studies, this paper proposes a new software defect prediction model. Representative features are automatically extracted from unmarked condition monitoring data in an unsupervised manner by a stacked denoising auto-encoder (SDAE). By stacking the trained denoising auto-encoder, the deep neural network (DNN) is constructed to perform intelligent defect diagnosis after fine-tuning the model with several available marker data. In this approach, a large number of easily accessible unmarked status monitoring data is utilized to learn useful and robust features. Only a small amount of tag data is required, which is advantageous in practical applications. In addition, after further fine-tuning the trained DNN, the software defect prediction can be correctly classified by the proposed method. The rest of the paper is organized as follows. In Sect. 2, the prediction model based on SDAE are introduced in detail. In Sect. 3, the experiment on open source dataset MDP is discussed and the results are also displayed. Summary is made in Sect. 4.

2 Software Defect Prediction Model Based on SDAE

2.1 Data Preprocessing Method Based on SMOTE Algorithm

In the software system, the number of high-risk modules is relatively small, and the dataset is imbalanced dataset, and the class that is a small number is called a minority class. To solve this problem, Chawla et al. [9] proposed a synthetic minority over-sampling technique (SMOTE). The specific steps of the algorithm are as follows:

1. The training sample is T, and D is a minority sample set in T. For each minority class sample $x_i \in D$, calculate the Euclidean distance of x_i to other minority samples $x_j (j \neq i)$ and find the K neighbors of x_i according to the distance, denoted as $x_{i(near)}, near \in \{1, 2, \ldots k\}$.
2. Randomly select a sample $x_{i(j)} \in x_{i(near)}$ and generate a new minority sample according to the following formula: $x_{i1} = x_i + rand(0, 1) * (x_{i(j)} - x_i)$, where $rand(0, 1)$ represents a random number between 0 and 1.
3. Repeat step 2 according to the over-sampling rate M, then add all generated minority samples to D.

2.2 Stacked Denoising Auto-encoder

We propose a software defect prediction model that uses a deep neural network (DNN) based on stacked denoising auto-encoder. Representative features are learned by applying the denoising auto-encoder to the unlabeled data in an unsupervised manner. A DNN is then constructed and fine-tuned with just a few items of labelled data.

Auto-encoder. An auto-encoder is a three-layer neural network that tries to reconstruct the input at the output layer after being passed through an intermediate layer [10]. A sample auto-encoder is shown in Fig. 1 where it tries to learn a function $h_{W,b}(\mathrm{x}) \approx \mathrm{x}$, that is, the output layer is trying to be equal to the input layer. W, b correspond to the weight matrix and bias of the input respectively. There is no limit to the hidden layer's size, and the circle of "+1" means the biases.

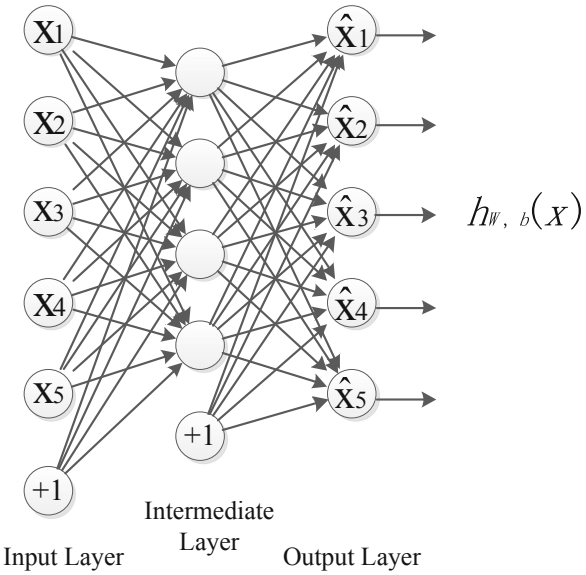

Fig. 1. Structure of an auto-encoder network.

The auto-encoder network's goal is to make the output is close to the input so that the output is the reconstruction of the input. In this way, the hidden layer can retained much of the information from the input data and may be a good expression of the input. Given a training set, with training examples, the cost function is formulated as:

$$\begin{aligned} \mathrm{J(W,b)} &= \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(w_{ji}^{(l)})^2 \\ &= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}||h_{w,b}(x^{(i)}) - y(i)||^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(w_{ji}^{(l)})^2 \end{aligned} \tag{1}$$

The first term in the formula is the mean square error, and the second term is the regularization, also called the weight decay, which is used to control the magnitude of the weight decay, thereby avoiding overfitting. $J\left(W,b;x^{(i)},y^{(i)}\right)$ is the square error relative to a single sample, and $\mathrm{J(W,b)}$ is the loss function of the overall sample. λ is the coefficient of weight penalty.

In order to train the neural network, it is first necessary to initialize $W_{ji}^{(l)}$ and $b_i^{(l)}$ with an initial value close to 0, and then use the gradient descent algorithm to optimize the cost function J(W, b).

The weights and bias of the network are updated using backpropagation algorithm as defined in [11]. Each iteration of the gradient descent algorithm updates the parameters W and b as follows:

$$W_{ji}^{(l)} = W_{ji}^{(l)} - \alpha \frac{\partial}{\partial W_{ji}^{(l)}} J(\mathrm{W,b}) \tag{2}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b) \tag{3}$$

Where α is the learning rate.

$$\frac{\partial}{\partial w_{ji}^{(l)}} J(\mathrm{w,b}) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial w_{ji}^{(l)}} J\left(w, b; x^{(i)}, y^{(i)}\right) + \lambda W_{ji}^{(l)} \tag{4}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_i^{(l)}} J(W, b; x^{(i)}, y^{(i)}) \tag{5}$$

Denoising Auto-encoder. The denoising auto-encoder (DAE) is an improvement based on the ordinary auto-encoder, and its purpose is not to reduce noise, but to learn more features that are robust. The structure of DAE is showed in Fig. 2. First, the initial input x is corrupted into $\tilde{x}$ through the stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$. The mapping q_D can be described as: a fraction of the elements of data x chosen randomly is forced to 0. After corrupted input $\tilde{x}$ is got, the steps of encoder and decoder can be calculated as the ordinary auto-encoder to get the hidden output $y = f_\theta(\tilde{x})$ and reconstruction output $\hat{x} = g_{\theta'}(y)$. The cost function is the squared error loss $L_2(x - \hat{x}) = \|x - \hat{x}\|^2$, which is minimized by updating the parameters.

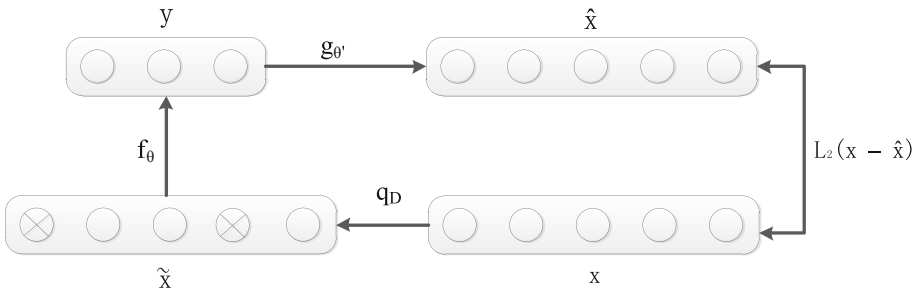

Fig. 2. Structure of a denoising auto-encoder network.

The difference between the DNA and the ordinary auto-encoder is that $\hat{x}$ here is a deterministic function of $\hat{x}$ rather than x. It thus encourages the learning of a cleverer mapping than the identity: one that extracts features useful for denoising [12]. The layer-wise procedure is the same as the ordinary auto-encoder. The input corruption is used only for the training of each layer to learn useful representations. After the mapping f_θ is learnt, uncorrupted inputs are used to produce a representation that will serve as the clean input to the following layer. Different types of corruption processes may be considered such as additive isotropic Gaussian noise, salt-and-pepper noise and masking noise [13]. In this way, DAE utilizes the denoising as a training criterion to extract stable and robust features.

Stacked Denoising Auto-encoders. The stacked denoising auto-encoder network stacks multiple denoising auto-encoder networks to form a deep network model. The output of the previous hidden layer is used as the input of the latter denoising auto-encoder. The learning of denoising auto-encoder is performed layer by layer until the last hidden layer, and the output of this layer is the learned high-level output features. Note that the input corruption is only used for the initial denoising-training of each individual layer which acts as feature extractors. After the parameters are learned, the no corruption input is applied to produce the features that will be the clean input of the next layer [14]. The stacking procedure of denoising auto-encoders is shown in Fig. 3.

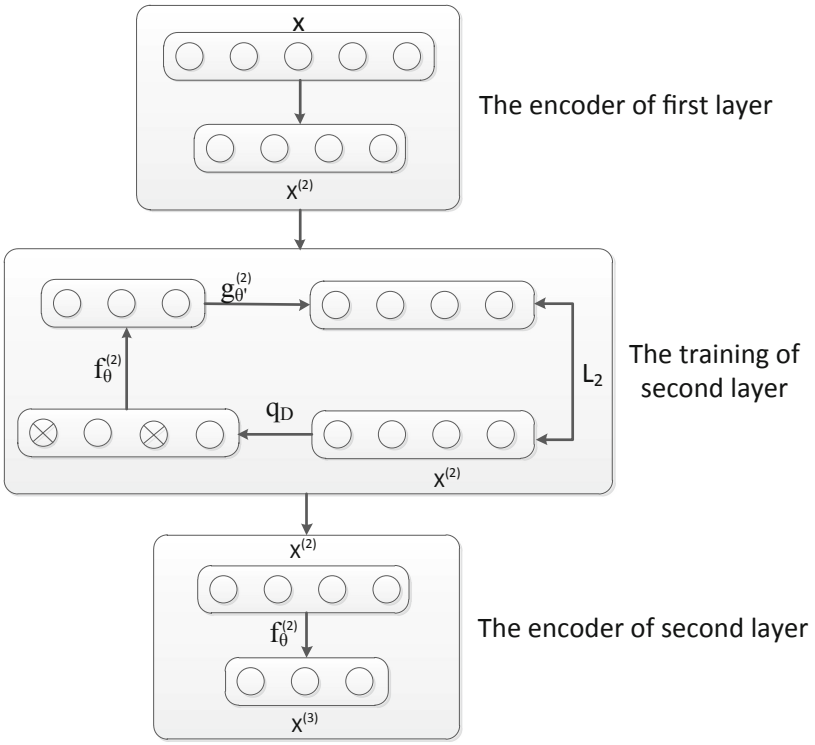

Fig. 3. Structure of a stacked denoising auto-encoder network.

Once the stack denoising auto-encoders (SDAE) is built, we can get the highest features from the last hidden layer. In order to get more discriminative features, a stand-alone supervised algorithm of Softmax regression is added on the top of the stack as illustrated in Fig. 4. The parameters of all layers then can be optimized using the stochastic gradient descent. And the classification accuracy result is able to be got from the Softmax model [15], too.

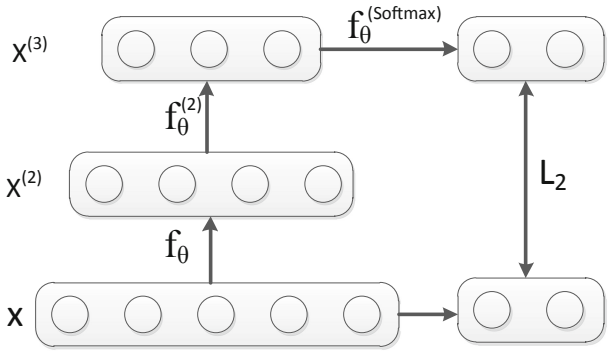

Fig. 4. Structure of softmax layer.

Step 1: Data pre-processing. Standardize the acquired data and then balance the data with the SMOTE algorithm.
Step 2: Initialization of the DNN. A DNN with N-hidden layers is initialized with random parameters.
Step 3: Unsupervised feature learning. Denoising autoencoder is applied here to learn representative features from the unlabeled data.
Step 4: Supervised fine-tuning of the model. After Step 3, a softmax layer is added on top of the DNN. Labelled condition data is used to fine-tune the parameters of the DNN by stochastic gradient descent.
Step 5: Fault diagnosis with the trained DNN. With the trained DNN, fault prediction can be carried out.

3 Experiments and Results

NASA dataset is widely used in the research of defect prediction technology. The data is derived from the source code of McCabe and Halstead attributes. We use NASA's MDP public data set, and select JM1, MC1 and PC5 three sub-data sets with a large amount of data. These three data sets are all imbalanced data sets, and the defect data is a minority class, as shown in the Table 1.

We rely on AMD Opteron (tm) Processor 6320, RAM 32 GB, 64-bit operating system, python 3.5.4 and so on. In the typical defect prediction technology research, the estimation results are usually evaluated using the confusion matrix related evaluation performance indicators, such as accuracy, precision, recall, F-measure and so on. The confusion matrix includes the correct positive case (TP), the wrong positive case

Table 1. Dataset summary.

Dataset	Number of samples	Number of attributes	Number of samples in the majority class	Number of samples in the minority class	Proportion of minority samples
JM1	7782	21	6110	1672	21.5%
MC1	1988	38	1942	46	2.3%
PC5	1711	38	1240	471	27.5%

(FP), the wrong negative case (FN), and the correct negative case (TN). TP is the number of modules predicted to contain defects and are actually defects. FP is the number of modules predicted to contain defects and are actually free of defects. FN is the number of modules predicted to contain defects and are actually free of defects. FN is the number of modules predicted to be free of defects and are actually defective, and TN is the number of modules predicted to be free of defects and that do not actually contain defects. The specific definition is as follows.

$$\text{Accuracy} = \frac{TN + TP}{TN + TP + FN + FP} \tag{6}$$

$$\text{Precision} = \frac{TP}{TP + FN} \tag{7}$$

$$\text{Recall} = \frac{TP}{TP + FP} \tag{8}$$

$$\text{F} - \text{measure} = \frac{(\alpha^2 + 1)Precision * Recall}{\alpha^2(\text{Precision} + \text{Recall})} \tag{9}$$

Usually, $\alpha = 1$.

3.1 Data Preprocessing Based on SMOTE Algorithm

For imbalanced data sets, it is unreasonable to use only the confusion matrix-based metrics to evaluate the performance of the prediction model. For example, when the minority class proportion is less than 1%, even if all the minority samples are divided into the majority class, the total precision can still reach 99%, but such a model has no practical significance, and its classification accuracy rate for the minority is 0. Therefore, we will select the F-measure of the minority class and the overall AUC value as the evaluation indicators. The prediction of minority samples, that is, the defect samples, is what we need to pay attention to. F-measure can comprehensively consider the recall and precision of the samples. The ROC (Receiver Operating Characteristic curve) curve is a coordinate pattern analysis tool that can describe the classification performance of the classifier at different discriminant thresholds. The abscissa is the FP and the ordinate is TP. In practical applications, the performance of the classifier is generally evaluated by replacing the ROC curve with the AUC (Area Under the ROC Curve). We use SVM and KNN algorithms to classify data separately.

The Tables 2 and 3 lists the F-measure values of minority class and AUC values on the original and processed datasets using the SVM and KNN classifiers.

It can be seen from the table that after the imbalanced data is processed by the SMOTE algorithm, the classification effect of the minority samples is significantly improved, and the classification effect of the overall data is also improved.

Table 2. Metrics for the original dataset and the processed dataset using SVM.

Dataset		F-measure	AUC
JM1	Original dataset	0.23	0.56
	Processed dataset	0.60	0.65
MC1	Original dataset	0.00	0.50
	Processed dataset	0.76	0.75
PC5	Original dataset	0.11	0.52
	Processed dataset	0.61	0.62

Table 3. Metrics for the original dataset and the processed dataset using KNN.

Dataset		F-measure	AUC
JM1	Original dataset	0.23	0.55
	Processed dataset	0.75	0.72
MC1	Original dataset	0.00	0.50
	Processed dataset	0.87	0.86
PC5	Original dataset	0.33	0.56
	Processed dataset	0.69	0.68

3.2 Data Prediction Based on SDAE

We predict and classify the data processed by the SMOTE algorithm. Through experiments, we can determine that when the number of hidden layers in SDAE is 2, the number of cells per layer is [6, 11], and the learning rate is 0.1, the model performance is optimal. In order to verify the superiority of the proposed method with respect to the traditional classification method, the results were compared with SVM and KNN. The experimental results were evaluated by accuracy, recall, precision and F-measure. The detailed software defect prediction results were shown in the Table 4.

Obviously, when the SVM, KNN, and SDAE are measured using the Accuracy, Recall, Precision, and F-Measure metrics on the three software defect sets of JM1, MC1, and PC5, the metrics of SDAE are generally compared. It can be seen that the prediction performance has improved significantly. The reason why SDAE prediction and classification ability is stronger than SVM and KNN is that its deep nonlinear network abstracts the original data layer by layer, and obtains features that are more capable of describing the essence of the object and easy to classify. Denoising pre-training is performed for each layer of DAE. The robustness of the extracted features is further enhanced, and the spatial features of the data are more fully explored.

Table 4. Experimental results.

Dataset	Model	Accuracy	Precision	Recall	F-measure
JM1	SVM	0.65	0..65	0.65	0.64
	KNN	0.72	0.73	0.72	0.72
	SDAE	0.82	0.80	0.83	0.81
MC1	SVM	0.76	0.76	0.76	0.76
	KNN	0.86	0.87	0.86	0.86
	SDAE	0.87	0.89	0.86	0.87
PC5	SVM	0.66	0.66	0.66	0.66
	KNN	0.67	0.68	0.67	0.67
	SDAE	0.81	0.74	0.79	0.81

4 Summary

Aiming at the limitations of existing software defect prediction models that can't dig deeper into data features, this paper proposes to use the stack denoising auto-encoder in deep learning to learn the data of software defect datasets, mine data features, and build software defect prediction model. The actual data and comparative experimental results show that the SDAE model used in this paper has high predictability and practicability for software defects. In the following work, the differences between other non-neural network classification algorithms and SDAE in software defect prediction applications will be further studied, and the various parameter adjustment methods and structures of SDAE will be improved.

References

1. National Institute of Standards and Technology (NIST): Software Errors Cost U.S. Economy $59.5 Billion Annually. http://www.abeacha.com/NIST_press_release_bugs_cost.htm
2. Briand, L.C., Brasili, V.R., Hetmanski, C.J.: Developing interpretable models with optimized set reduction for identifying high-risk software components. IEEE Trans. Software Eng. **19**(11), 1028–1044 (1993)
3. Evett, M., Khoshgoftar, T., Chien, P.D., et al.: GP-based software quality prediction. In: Proceedings of the Third Annual Conference Genetic Programming, pp. 60–65 (1998)
4. Pizzi, N.J., Summers, R., Pedrycz, W.: Software quality prediction using median-adjusted class labels. In: Proceedings: International Joint Conference on Neural Networks, vol. 3, pp. 2405–2409 (2002)
5. Mahaweerawat, A., Sophasathit, P., Lursinsap, C.: Software fault prediction using fuzzy clustering and radial basis function network. In: International Conference on Intelligent Technologies, pp. 304–313 (2002)
6. Jindal, R., Malhotra, R., Jain, A.: Software defect prediction using neural networks. In: IEEE 3rd International Conference on Reliability, Infocom Technologies and Optimization (ICRITO) (Trends and Future Directions), pp. 1–6 (2014)

7. Yang, X., Lo, D., Xia, X., et al.: Deep learning for just-in-time defect prediction. In: IEEE International Conference on Software Quality, Reliability and Security (QRS), pp. 17–26 (2015)
8. Phan, A.V., Nguyen, M.L., Bui, L.T.: Convolutional neural networks over control flow graphs for software defect prediction. In: IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), vol. 1, pp. 45–52 (2017)
9. Chawla, N.V., Bowyer, K.W.: SMOTE: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 341–378 (2002)
10. Thirukovalluru, R., Dixit, S., Sevakula, R.K., et al.: Generating feature sets for fault diagnosis using denoising stacked auto-encoder. In: IEEE International Conference on Prognostics and Health Management, pp. 1–7 (2016)
11. Ng, A., Ngiam, J., Foo, C.Y., Mai, Y., Suen, C.: UFLDL tutorial. Accessed Mar 2016. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
12. Feng, X., Zhang, Y., Glass, J.: Speech feature denoising and dereverberation via deep autoencoders for noisy reverberant speech recognition. In: IEEE International Conference on Acoustics, pp. 1759–1763 (2014)
13. Vincent, P., Larochelle, H., Bengio, Y., et al.: Extracting and composing robust features with denoising autoencoders. In: ACM International Conference on Machine Learning, pp. 1096–1103 (2008)
14. Xiao, N., Liu, D., Luo, A., et al.: Adaptive feature extraction based on Stacked Denoising Auto-encoders for asynchronous motor fault diagnosis. In: IEEE International Congress on Image and Signal Processing, Biomedical Engineering and Informatics, pp. 854–859 (2017)
15. Ma, J., Lu, C., Zhang, W., et al.: Health assessment and fault diagnosis for centrifugal pumps using Softmax regression. J. Vibroengineering **16**(3), 1464–1474 (2014)