Salekshahrezaee *et al. J Big Data* (2021) 8:57
https://doi.org/10.1186/s40537-021-00447-5

Journal of Big Data

RESEARCH Open Access

A reconstruction error-based framework for label noise detection

Zahra Salekshahrezaee, Joffrey L. Leevy* and Taghi M. Khoshgoftaar

*Correspondence:
jleevy2017@fau.edu
Florida Atlantic University,
777 Glades Road, Boca Raton,
FL 33431, USA

Abstract

Label noise is an important data quality issue that negatively impacts machine learning algorithms. For example, label noise has been shown to increase the number of instances required to train effective predictive models. It has also been shown to increase model complexity and decrease model interpretability. In addition, label noise can cause the classification results of a learner to be poor. In this paper, we detect label noise with three unsupervised learners, namely *principal component analysis* (PCA), *independent component analysis* (ICA), and autoencoders. We evaluate these three learners on a credit card fraud dataset using multiple noise levels, and then compare results to the traditional Tomek links noise filter. Our binary classification approach, which considers label noise instances as anomalies, uniquely uses reconstruction errors for noisy data in order to identify and filter label noise. For detecting noisy instances, we discovered that the autoencoder algorithm was the top performer (highest recall score of 0.90), while Tomek links performed the worst (highest recall score of 0.62).

Keywords: Label noise, Autoencoder, PCA, ICA, Tomek links

Introduction

Classification involves predicting the class of a new sample by using a model derived from training data. A comprehensive and accurate collection of data is essential for supervised learning and classification algorithms [1]. In a labeled dataset, each sample (also known as an instance) is associated with an observed label. This label often corresponds to the real class of the sample (also known as the true class). Label noise, or class noise, exists when the real class is different from the observed class, e.g. a majority class (negative) label wrongly assigned to a positive instance. For example, an innocent person in a large city could be wrongly tagged as a crime suspect by a machine learning algorithm. Models trained on datasets with high levels of label noise will not generalize well to new data [2, 3].

Sometimes, a data sample may have one or more attributes that are incorrect or corrupt. This condition is known as feature noise (or attribute noise) [4, 5]. Label noise has been shown to be potentially more harmful than feature noise [6]. The reason is that a dataset usually has more than one feature, and the value and weight of each feature may be different for training purposes. However, for each dataset there

is typically one class label, and this label often has a significant effect on classification performance [1, 6].

There are several common issues that can cause label noise. It may stem from inadequate information provided to the expert that, in turn, results in inaccurate labeling [1, 7]. Label noise may also be caused by the limited specificity of classification [7], which can decrease the true accuracy of the classification. Thirdly, label noise may be a consequence of the unreliable quality of information [8]. To alleviate these issues, various manual and/or automated services are used to assist with classification [1]. Amazon Mechanical Turk, made popular with its inexpensive and user-friendly labeling processes, provides one such service [9].

Label noise can also come from data encoding or network connectivity problems [7, 10]. Real-world databases are believed to contain about 5% encoding errors in their data [6]. As a result of these errors, there is a certain amount of unavoidable label noise [11]. Unfortunately, this necessitates the inclusion of either additional features or an increased training data size to compensate for this inherent label noise [7].

Our work is grounded on the concept that label noise data points are likely to be anomalies in the class indicated by the corresponding label [8, 12]. Stated otherwise, after a feature space is defined for a binary class, there will be some instances in an assigned class that belong to the other class due to label noise. Our contribution involves the novel approach of treating reconstruction error as a mapping technique in order to detect label noise. This error is the difference in *mean square error* (MSE) between the input vector and the reconstructed output produced.

Our machine learning models are trained by minimizing reconstruction error. Three unsupervised learning algorithms were used in this study: *principal component analysis* (PCA), *independent component analysis* (ICA) and autoencoders. They have previously been used by other researchers to detect anomalies in datasets [8, 10, 13]. We compared these unsupervised methods to Tomek links, a traditional label noise filter technique for removing data capable of impairing classification performance [14, 15].

In this study, our autoencoder model produced the best scores for all three metrics used: *Area Under the Receiver Operating Characteristic Curve* (AUC), recall, and *False Negative Rate* (FNR). Our results also showed that the Tomek links algorithm was the worst performer. For a given metric, it was generally observed that for each algorithm, the highest noise level yielded the best score, and the lowest noise level yielded the worst score.

The remainder of this paper is organized as follows: Section "Related Work" provides an overview of literature related to label noise detection; Section "Background" provides relevant information on all algorithms used in this study (PCA, ICA, autoencoders, Tomek links); Section "Methodology" describes the training and testing of the unsupervised learners, the levels of noise used, and the metrics used; Section "Results and Discussion" presents and discusses our empirical results; and Section "Conclusion" concludes our paper with a summary of the work presented, along with suggestions for related future work.

Related work

There are two main techniques that address label noise. The first approach filters noisy instances and cleans the data, while the second develops methods that are robust when facing label noise during the modeling process.

In the filtering approach, noisy instances are removed or relabeled to get a cleaner dataset [16]. One of the earlier works on noise filtering is *edited nearest neighbor* (ENN) by Wilson [17]. This algorithm removes an instance in a training set if the point does not agree with the majority of its *k-nearest neighbors* (k-NNs). Editing of the instances is done using the three-nearest neighbor rule followed by classification using the single-nearest neighbor rule. Tomek *et al.* [14] later suggested an extension of ENN. In their approach, they apply the nearest neighbor rule k times. For each execution, the nearest neighbor rule changes the number of neighbors considered between 1 and k. If one instance is misclassified by the rule, it is removed. In another related study, Khoshgoftaar *et al.* [18] introduced iterative partitioning filtering. This approach eliminates noisy instances in multiple iterations until a stopping criteria is met. The iterative process ceases if, for a number of consecutive iterations, the number of noisy examples found in each of these iterations is less than a proportion of the size of the original training dataset. Each iteration splits the current training dataset into subsets of equal sizes, followed by the creation of a C4.5 decision tree model for each of these subsets. A voting strategy is then used (consensus or majority voting) to identify noisy instances. Sáez *et al.* [19] performed a more recent study, where they introduced an iterative class noise filter based on the fusion of classifiers. Their technique eliminates noisy instances in multiple iterations. It is based on three major noise filter paradigms: ensemble-based filtering, iterative filtering, and metric-based filtering. For each iteration, the first step eliminates part of the existing noise in the current iteration to reduce its effect in the subsequent steps. Our study also adopts a filtering approach. However, we focus on unsupervised learning, which is ideal for studying raw data.

With regard to the development of robust methods, the aim is to render mislabeled instances less destructive to the model. Strategies such as robust loss functions [20] that prevent overfitting can be implemented during training to make the classifier less responsive to noisy data. In addition, several studies on label noise detection have explored ensemble learning, which is only suitable for comparatively simple cases [21]. On the other hand, several approaches are explicitly designed for label noise problems using various robust optimization methods [22]. We point out that these are primarily supervised approaches [23]. Zhang and Tan [24] used a reconstruction error minimization framework to further filter and relabel the mislabeled data. Their results, which are based on the MNIST dataset [25], show that their proposed method could significantly reduce the false positive outlier detection rate and improve both data cleaning and classification quality. The authors follow an unsupervised approach just like we did. However, they use a dataset of images.

Background

PCA

In machine learning, PCA is an exploratory data analysis method that reveals the inner structure of the data and explains variance. To create a more compact feature space, the technique identifies correlated variables and effects a transformation that best reflects the differences from the input [26]. The application of PCA results in m principal components. If the amount of variance captured by the first r-principal components represents the majority of the variance in the data, the data can be mapped to the reduced r-dimensional space represented by the first r-principal components.

PCA is typically used for dimension reduction. In this study, however, we use it for label noise detection by applying the subspace method [27]. The subspace method works by dividing the principal axes into two sets representing normal and anomalous data variations. Any data instance represented by a row in the dataset can be defined as $y = \hat{y} + \tilde{y}$ by representing it as normal ($\hat{y}$) and anomalous subspace ($\tilde{y}$). The concept of subspace anomaly detection uses an anomalous shift in the feature correlations in a vector function to represent an increment in the projection of that data point to an anomalous subspace. The magnitude of $\tilde{y}$ is higher than what is seen among normal instances [28]. To determine the magnitude of the projection of each instance to an anomalous subspace, we first examine the set of principal components in the normal subspace as columns of the matrix P of size $m \times r$.

$$\hat{y} = PP^T y = Cy \tag{1}$$

$$\tilde{y} = (1 - PP^T)y = \tilde{C}\tilde{y} \tag{2}$$

The $PP^T y$ matrix represents the linear projection to the normal subspace. $\tilde{C}$ represents a linear projection to an anomalous subspace, and abnormal variations $\tilde{y}$ represent an anomaly. The *square prediction error* (SPE) is used to monitor the changes in $\tilde{y}$. SPE is computed as shown in [29]:

$$SPE = \|\tilde{y}\|^2 \tag{3}$$

If an instance of SPE is more than a threshold, this instance is identified as anomalous, and in our case, it will be detected as an anomaly to the class that the instance was labeled as. Each new input is analyzed for anomalies. The anomaly detection algorithm combined with a normalized reconstruction error computes its projection on eigenvectors. The normalized error is used as the score for the anomaly, and we presuppose that the higher the error value, the more anomalous the instance [30, 31]. This intuitive assumption also holds true for the ICA and autoencoder algorithms.

ICA

ICA is a statistical technique for analyzing hidden factors (sources or features) from a collection of measurements or records. The algorithm is able to separate the sources according to the distribution of the data. To achieve the separation of mixed data into independent components, ICA exploits the independence of the sources. This technique

is useful for separating anomalies [32–34]. To formally describe the ICA model, consider $X = (x_1, x_2, ..., x_n)$ as the input vector and $S = (s_1, s_2, ..., s_p)$ as a random vector of latent mutually independent sources where $p \leq n$. The ICA model can be described as

$$X = AS$$

ICA involves the calculation of both A and S where only X is known, with A being a matrix $(n \times p)$. A is presumed to be fixed, but unknown, and finds a matrix W such that $S = WX$. ICA obtains S based on the fact that members of this vector are statistically independent and non-Gaussian random variables [13, 35]. Ideally W^{-1} should be equal to A. However, it differs from A due to noise and outliers in mixed results. ICA methods define transformations such that components derived from mixtures are as independent as possible by optimizing or decreasing any objective function (kurtosis, entropy, etc.).

Autoencoder

In the field of neural networks, the concept of autoencoders has been popular for decades, with the most typical applications being the reduction of dimensionality or feature learning. More recently, the autoencoder concept has been used to learn generative data models and identify data anomalies [10, 36]. An autoencoder is an artificial neural network used in an unsupervised fashion to learn data representation [37]. The simplest type of autoencoder is comparable to a simple *multilayer perceptron* (MLP), with an input layer, an output layer and one or several hidden layers connecting them. The output layer has the same number of nodes (neurons) as the input layer in order to recreate its inputs. An autoencoder can be linear or non-linear, depending on its activation function [38].

The purpose of an autoencoder is to learn a representation (encoding) while disregarding signal noise. The algorithm attempts to minimize the difference between the input and the output, instead of predicting the target value Y given inputs X. A reconstructing side is learned along with the reduction side, where the autoencoder attempts to produce a representation as similar as possible to its original input from the reduced encoding. The size of the hidden layer determines the size of the latent representation and consists of two main components: an encoder that maps the input to the code and a decoder that maps the code to the original input [39]. The autoencoder reconstructs normal data efficiently after training, whereas it struggles to do so with anomalous data that was not encountered during training [40]. For high dimensional data with class anomalies, non-linear autoencoders have shown superior classification performance [41]. In this study, we use non-linear autoencoders.

To detect label noise in a dataset, the autoencoder is trained with clean data and learns the normal behavior of this data. The weight and architecture of the autoencoder are adjusted to minimize the reconstruction error for clean instances of the class. For our training dataset, each record can be described as $Z = Z_1, Z_2, ..., Z_n$. In this research, we inject various levels of noise into the test portion of the dataset by changing the label of a subset of the instances from one class to another. Reconstructed data points can be described as $\hat{Z} = g(W_2 f(W_1 Z + b_1) + b_2)$ [24]. W_1 and W_2 represent the weight matrix, while b_1 and b_2 are biases for encoder and decoder, and $f(x)$ and $g(x)$ are tanh activation functions. The loss function based on reconstruction error is described as follows:

$$loss = \frac{1}{n}\sum_{i=1}^{n} |\hat{Z}_i - Z_i|^2$$

PCA vs Autoencoder

For PCA modeling, several algorithms are available. One such algorithm involves estimation by minimizing reconstruction error [32]. In this subsection, we compare a linear single-layer autoencoder with PCA (minimizing reconstruction error method) for simplicity. Ranjan's depiction [41] of a linear single-layer autoencoder is shown in Fig. 1.

As shown at the bottom of the diagram, the process of encoding is identical to the transformation of *principal components* (PCs). Likewise, the method of decoding is similar to the reconstruction of data from the principal scores. For both the autoencoder and PCA, model weights are calculated through the minimization of the reconstruction error.

Suppose we have a dataset with p features and an encoding layer of size k. With PCA, k denotes the number of PCs chosen. For both the autoencoder and PCA, dimension reduction occurs when $k < p$. An over-representative model exists when $k \geq p$, which indicates a reconstruction error of virtually zero.

In the encoding layer of the autoencoder in Fig. 1, a colored cell is a computing node with p weights denoted as W_{ij} where $i = 1, ..., k$ and $j = 1, ..., p$. For each encoding node in 1, ..., k, there is a p-dimensional weight vector. This is equivalent to an eigenvector in PCA. The encoding layer output for an autoencoder is given as $z = g(Wx) = Wx$, if the activation function g is linear, where x is the input and W is the weight matrix [41]. If $g(Wx)$ is linear, this is the equivalent of the principal scores in PCA.

For autoencoder and PCA reconstruction, the size of the decoding layer must be the size of the input data p. In a decoder, the data is reconstructed from the encodings

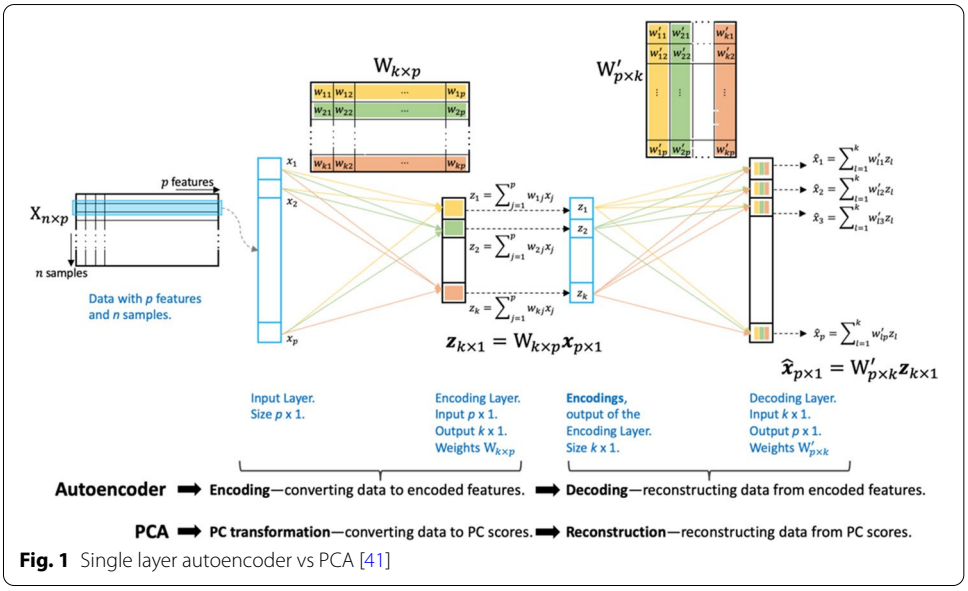

Fig. 1 Single layer autoencoder vs PCA [41]

and can be represented as $\hat{x} = g(\acute{W}z) = \acute{W}z$, if the activation function g is linear [41]. Similarly, for PCA, the data is reconstructed as $\hat{x} = W^T z$.

Tomek links

In this study, we also incorporated Tomek links, which is a traditional label noise detection technique. The Tomek links algorithm acts as a label noise filter and is denoted by pairs of instances. A Tomek link is a pair of data points x and y from different classes, such that, if d stands for the distance metric, there exists no example z such that $d(x, z)$ is lower than $d(x, y)$, or $d(y, z)$ is lower than $d(x, y)$. Hence, where the two examples x and y form a Tomek link, either one is noise or both are borderline [42]. These two examples are thus eliminated from the training data. To elaborate further, in a binary classification environment with classes 0 and 1, a Tomek link pair would have an instance of each class and would be nearest neighbors across the dataset [43]. These cross-class pairs are valuable in defining the class boundary [44]. Figure 2 by Argawal [45] shows an alignment of Tomek link pairs at the class boundary. It is important to note that the use of Tomek links for label noise detection does not involve the calculation of reconstruction error.

Methodology

Development of Models

We use a credit card fraud dataset [46] that was collected and analyzed through a research partnership between Worldline and the *Université Libre de Bruxelles* (ULB). The original dataset contains 248,807 instances and 31 columns (class label included). The credit card purchases were made by European cardholders in September 2013. There are 492 cases of fraud, or 0.172%, making the dataset highly imbalanced. A high imbalance exists within a dataset if the majority-to-minority class ratio ranges from 100:1 to 10,000:1 [47]. All but two of the independent variables in the fraud dataset are principal components obtained by PCA. The label is 1 in the event of fraud and 0 otherwise.

For training purposes, the original dataset was split into normal and fraud sub-datasets, and we subsequently trained the models using clean sub-datasets. We introduced

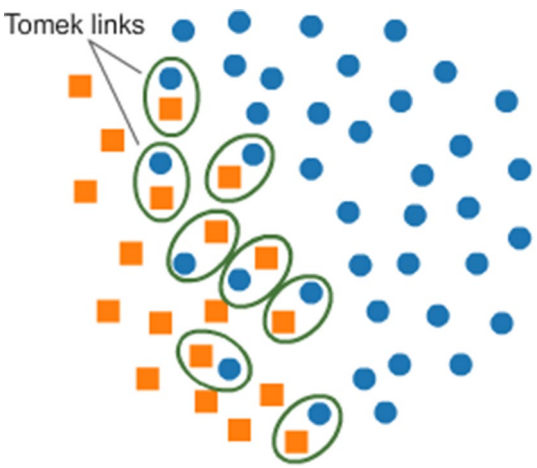

Fig. 2 Tomek link pairs [45]

noise by changing the labels for a number of instances in each class. Since the original dataset is imbalanced, the ratio of noisy to clean instances for the normal sub-dataset is also imbalanced. If, for example, we changed the label for all fraud cases from 1 to 0, it means we added 492 cases of label noise to the normal sub-dataset. In this case, the noise ratio is still less than 1% for this sub-dataset.

Our approach entailed *k*-fold *cross-validation* (CV), where the model is trained on *k*-1 folds each time and tested on the remaining fold. This is to ensure that as much data as possible are used in the classification. More specifically, we use stratified CV which attempts to ensure that each class is approximately equally represented across each fold [48]. In our study, we assigned a value of 4 to *k*: three folds for training and one fold for testing. We repeated the process of building and evaluating the models 10 times for each unsupervised learner and dataset. The use of repeats helps to reduce bias due to bad random draws when generating the samples. The final performance result is the average over all 10 repeats. The primary focus of our work is to detect fraudulent instances mislabeled as normal instances (minority class instances mislabeled as the majority class). For each step of the experiment, we swapped the label for a specific number of instances, up to 90% of instances of the minority class and a very small number of instances from the majority class. The number of noisy examples in the training results is specified as *noise level* (NL).

We used four levels of noise, NL = {10, 40, 70, 90}. Similar to the formula used in [5], the actual number of noisy examples is given as

$$\frac{NL}{100} \times P + \frac{NL}{10}$$

P is the number of instances in the minority class. $NL/100 * P$ corresponds to the number of noisy instances injected into the negative class, while $NL/10$ corresponds to the relatively small number of noisy instances injected into the positive class. For example, when NL is 40, the number of noisy examples is calculated as $40/100 * 492 + 4 = 201$.

The PCA configuration used in this work has the same number of principal components as the number of original features from our dataset. The inverse transform function from Scikit-Learn recreated the original transactions from the key components produced [49].

For our ICA algorithm, we implemented the sklearn.decomposition.FastICA function from Scikit-learn [50]. The FastICA algorithm is an efficient and popular variant of ICA. Based on best performance during preliminary experimentation, we set the algorithm on 'parallel' mode, and set the max_iteration count to 200. Additionally, we also set the number of components equal to the input vector dimension size [50].

Our autoencoder was implemented in Keras using a TensorFlow backend [51]. Through preliminary experimentation, the best parameters were selected. The model contained nine densely connected hidden layers, along with the tanh activation function and L1 regularization. The input and output layers had the same dimensions as the number of features in the original dataset.

To identify an error threshold for each learner that discriminates between normal and anomalous behavior, we examined the distributions of the reconstruction errors. We then chose the *n*th percentile of the error distribution of the normal dataset [10].

If, for example, $n = 95$, this means that 95% of the errors in the normal dataset are smaller than a value (v). Hence, any data point fed to a specific learner that generates an error greater than v would be classified as anomalous. Our analysis showed that the best outcomes were achieved at higher thresholds, i.e. $n \geq 93$.

Metrics

Our work records the confusion matrix (Table 1) for a binary classification problem, where the class of interest is usually the minority class and the opposite class is the majority class, i.e. positives and negatives, respectively. A related list of simple performance metrics [52] is explained as follows:

- *True Positive* (TP) is the number of positive samples correctly identified as positive.
- *True Negative* (TN) is the number of negative samples correctly identified as negative.
- *False Positive* (FP), also known as Type I error, is the number of negative instances incorrectly identified as positive.
- *False Negative* (FN), also known as Type II error, is the number of positive instances incorrectly identified as negative.

Based on these fundamental metrics, other performance metrics are derived as follows:

- *Recall*, also known as *True Positive Rate* (TPR) or sensitivity, is equal to $TP/(TP + FN)$.
- *Specificity*, also known as *True Negative Rate* (TNR), is equal to $TN/(TN + FP)$.
- *Fall-out*, also known as *False Positive Rate* (FPR), is equal to $FP/(TN + FP)$.
- *Miss Rate*, also known as FNR, is equal to $FN/(TP + FN)$.
- AUC graphically shows recall versus (1-specificity), or TPR vs FPR, across all classifier decision thresholds [53]. From this curve, the AUC obtained is a single value that ranges from 0 to 1, with a perfect classifier having a value of 1.
  In our study, we use more than one performance metric (recall, FNR, AUC). This strategy allows us to better understand the challenge of evaluating the machine learning algorithms with highly imbalanced data.

Table 1 Confusion matrix

	Predicted Class	
	Positive	Negative
Actual class		
Positive	True Positive (TP)	False Negative (FN) (Type II error)
Negative	False Positive (FP) (Type I error)	True Negative (TN)

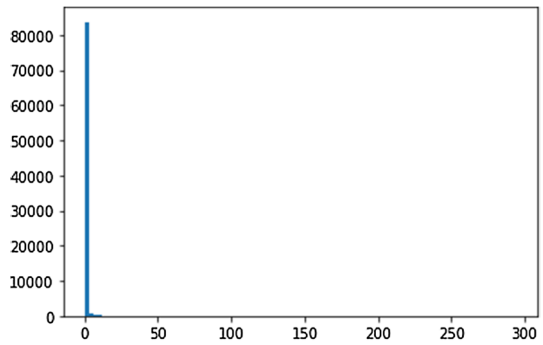

Fig. 3 Autoencoder—Number of instances vs reconstruction error value for clean data points

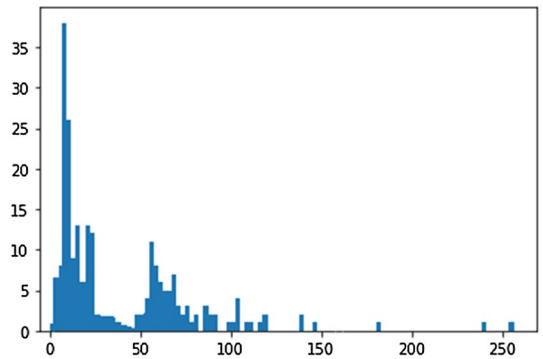

Fig. 4 Autoencoder—Number of instances vs reconstruction error value for noisy data points

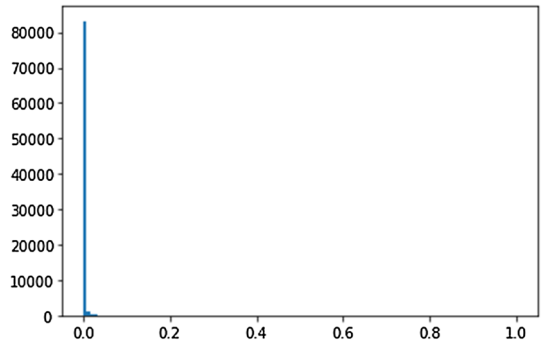

Fig. 5 ICA—Number of instances vs reconstruction error value for clean data points

Results and discussion

We expect to observe higher reconstruction errors for the mislabeled instances. Figures 3, 4, 5, 6, 7 and 8 are histograms that show error distribution for both clean and noisy data points for a noise level of 90, which corresponds to 452 noisy instances. These graphs plot the number of instances against reconstruction error. Tomek links are excluded for these figures because this algorithm does not rely on reconstruction error calculations for label noise detection. As indicated in Figs. 3, 4, 5, 6, 7 and 8, the distribution of reconstruction error is higher for the noisy instances than the clean instances.

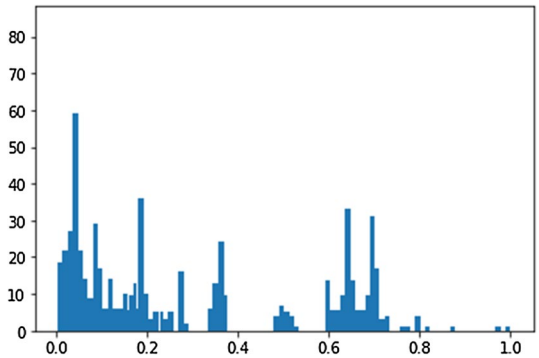

Fig. 6 ICA—Number of instances vs reconstruction error value for noisy data points

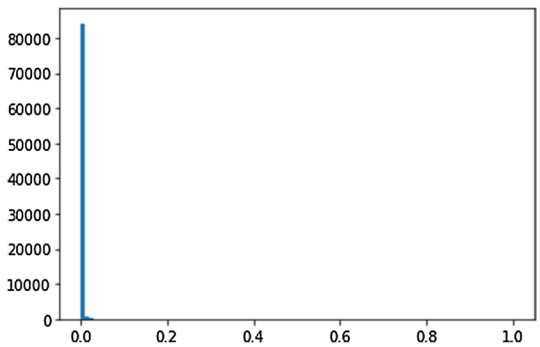

Fig. 7 PCA—Number of instances vs reconstruction error value for clean data points

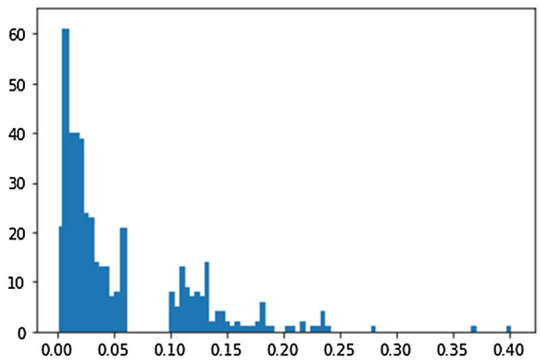

Fig. 8 PCA—Number of instances vs reconstruction error value for noisy data points

Performance results for label noise detection are presented as average scores in Table 2. Each value is the mean of 40 repetitions (10 runs per experiment x 4-fold cross validation). The best performance score within each column for each algorithm is in italic type.

When comparing the same noise levels (same number of noisy instances) among the algorithms for each metric in Table 2, we see that the autoencoder has the best scores. Overall, the highest scores for AUC (0.96) and recall (0.90) were obtained with the autoencoder, while the lowest FNR score (0.10) was also obtained with this model.

Table 2 Label noise detection results

Algorithm	Noisy instances	AUC	FNR	Recall
Autoencoder	452	*0.96*	*0.10*	*0.90*
	352	0.95	0.12	0.88
	201	0.95	0.14	0.86
	51	0.94	0.18	0.82
ICA	452	*0.94*	*0.14*	*0.84*
	352	0.93	0.15	0.82
	201	0.92	0.17	0.83
	51	0.91	0.18	0.81
PCA	452	*0.94*	*0.13*	*0.83*
	352	0.93	0.15	0.81
	201	0.93	0.17	0.81
	51	0.90	0.18	0.80
Tomek links	452	*0.90*	*0.50*	0.54
	352	0.89	*0.50*	0.53
	201	*0.90*	0.53	0.55
	51	0.88	0.60	*0.62*

Table 3 Two-factor ANOVA for recall

Factor	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithms	3	8.840	2.9466	7066.09	<2e-16
Noise level	3	0.081	0.0270	64.77	<2e-16
Interaction	9	0.252	0.0280	67.16	<2e-16
Residuals	624	0.260	0.0004		

Most tellingly, the Tomek links algorithm appears to be the worst performer. In general, we observe that for each algorithm, per metric, the highest noise level yielded the best score, and the lowest noise level yielded the worst score. The autoencoder model, for example, obtained its highest recall score of 0.90 for 452 noisy instances, and its lowest recall score of 0.82 for 51 noisy instances. This occurs because the algorithms become more efficient at detecting the class of interest (noisy instances) as the noise level increases. For a machine learning algorithm, the task of detecting a target class containing an extremely low number of instances is comparable to searching for a needle in a haystack [54]. We note that a few cases in Table 2 go against the norm, such as the best recall score for Tomek links (0.62), which is associated with the lowest noise level of 51 instances.

From a statistical point of view, it is beneficial to understand the significance of the performance scores in Table 2. Hence, we conduct *ANalysis Of VAriance* (ANOVA) tests to determine the impact of the algorithms on performance in terms of recall, FNR and AUC. ANOVA is a statistical test determining whether there is significant difference between group means [55]. We use a 95% ($\alpha = 0.05$) confidence level for the ANOVA tests shown in Tables 3, 4 and 5, where *Df* is the degrees of freedom, *Sum Sq* is the sum of squares, *Mean Sq* is the mean sum of squares, *F value* is the F-statistic, and *Pr(>F)* is the *p*-value.

Table 4 Two-factor ANOVA for FNR

Factor	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithms	3	17.833	5.944	13230.69	<2e-16
Noise level	3	0.522	0.174	387.55	<2e-16
Interaction	9	0.114	0.013	28.14	<2e-16
Residuals	624	0.280	0.000		

Table 5 Two-factor ANOVA for AUC

Factor	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Algorithms	3	1.0855	0.3618	1366.760	<2e-16
Noise level	3	0.1539	0.0513	193.787	<2e-16
Interaction	9	0.0041	0.0005	1.733	0.0783
Residuals	624	0.1652	0.0003		

Table 6 Tukey's HSD results

Metric	Autoencoder	ICA	PCA	Tomek links
Recall	a	b	c	d
FNR	a	b	b	c
AUC	a	b	c	d

The algorithms are the factor of concern for our study. Because this factor has a *p*-value of less than 0.05 in all our ANOVA tables, this indicates that algorithms are a significant factor for all our metrics. Therefore, we perform Tukey's *Honestly Significant Difference* (HSD) tests to determine which groups are significantly different from each other. Letter groups assigned via the Tukey method indicate similarity or significant differences in performance results within a factor [56]. For all metrics, the letter grades reported in Table 6 corroborate our earlier findings. Results show that autoencoders are in group 'a', which is the top group, while Tomek links are in groups 'c' and 'd', which are the worst-performing groups.

In essence, this study shows that our autoencoder model can efficiently detect label noise in imbalanced large data. The model can also be successfully used to detect label noise in imbalanced big data. This is because non-linear autoencoders are capable of modeling complex functions by compressing data to lower dimensions. Detecting label noise in big data is important due to the unique challenges posed by this type of data. Such challenges arise from the volume, variety, velocity, variability, complexity, and value of big data [47].

It should be noted that the computational cost of training the encoder is relatively low, since for each class, the autoencoder needs to be trained only once. In addition, the comparatively high scores for recall and AUC, as well as the comparatively low scores for FNR, are a testament to the robust nature of our autoencoder model.

Conclusion

In this paper, we propose a novel and effective method to deal with the label noise problem. Our method considers label noise as anomalous instances in the class where they have been mislabeled. As a first step, we trained unsupervised learners (autoencoders, ICA, PCA) using clean data for each class. The trained models were then used to reconstruct unseen portions of data containing different levels of label noise, with higher reconstruction errors being produced for instances that were incongruent with their assigned class. The best detection rates for label noise were obtained by the autoencoders. Performance-wise, when the unsupervised learners were compared with Tomek links, we discovered that this traditional algorithm was the worst performer.

Our proposed model is a potential solution for detecting label noise in imbalanced big data. The first-rate performance of our non-linear autoencoder is due to the low computational costs involved in training and the algorithm's ability to successfully model complex functions through dimension reduction. Future work will include additional performance metrics and also datasets from different application domains.

Abbreviations

ANOVA: ANalysis Of VAriance; AUC: Area Under the Receiver Operating Characteristic Curve; CV: Cross-validation; ENN: Edited nearest neighbor; FN: False Negative; FNR: False Negative Rate; FP: False Positive; FPR: False Positive Rate; HSD: Honestly Significant Difference; ICA: Independent component analysis; *k*-NN: k-nearest neighbor; MLP: Multilayer perceptron; MSE: Mean square error; NL: Noise level; NSF: National Science Foundation; PC: Principal component; PCA: Principal component analysis; SPE: Square prediction error; TN: True Negative; TNR: True Negative Rate; TP: True Positive; TPR: True Positive Rate; ULB: Université Libre de Bruxelles.

Acknowledgements

We would like to thank the reviewers in the Data Mining and Machine Learning Laboratory at Florida Atlantic University. Additionally, we acknowledge partial support by the *National Science Foundation* (NSF) (CNS-1427536). Opinions, findings, conclusions, or recommendations in this paper are the authors' and do not reflect the views of the NSF.

Authors' contributions

ZS carried out the conception and design of the research, performed the implementation and experimentation, and drafted the manuscript. All authors provided feedback to ZS and helped shape the research. ZS and JLL prepared the manuscript. TMK introduced this topic to ZS and helped to complete and finalize this work. All authors read and approved the final manuscript.

Funding

Not applicable

Data availability

Not applicable

Declarations

Ethics approval and consent to participate

Not applicable

Consent for publication

Not applicable

Competing interests

The authors declare that they have no competing interests.

Received: 20 January 2021 Accepted: 7 April 2021
Published online: 15 April 2021

References

1. Angluin D, Laird P. Learning from noisy examples. Mach Learn. 1988;2(4):343–70.

2. Prati RC, Luengo J, Herrera F. Emerging topics and challenges of learning from noisy data in nonstandard classification: a survey beyond binary class noise. Knowl Inf Syst. 2019;60(1):63–97.
3. Pelletier C, Valero S, Inglada J, Champion N, Marais Sicre C, Dedieu G. Effect of training class label noise on classification performances for land cover mapping with satellite image time series. Remote Sens. 2017;9(2):173.
4. Van Hulse JD, Khoshgoftaar TM, Huang H. The pairwise attribute noise detection algorithm. Knowl Inf Syst. 2007;11(2):171–90.
5. Khoshgoftaar TM, Van Hulse J. Empirical case studies in attribute noise detection. IEEE Trans Syst Man Cybern C. 2009;39(4):379–88.
6. Maletic JI, Marcus A. Data cleansing: Beyond integrity analysis. In: Iq, pp. 200–209; 2000. Citeseer.
7. Wang D, Tan X. Robust distance metric learning via Bayesian inference. IEEE Trans Image Process. 2017;27(3):1542–53.
8. Patel AA. Hands-On Unsupervised Learning Using Python: How to Build Applied Machine Learning Solutions from Unlabeled Data. O'Reilly Media; 2019.
9. Raykar VC, Yu S, Zhao LH, Valadez GH, Florin C, Bogoni L, Moy L. Learning from crowds. J Mach Learn Res. 2010;11:4.
10. Borghesi A, Bartolini A, Lombardi M, Milano M, Benini L. Anomaly detection using autoencoders in high performance computing systems. Proc AAAI Conf Artif Intell. 2019;33:9428–33.
11. Frénay B, Verleysen M. Classification in the presence of label noise: a survey. IEEE Trans Neural Netw Learn Syst. 2013;25(5):845–69.
12. Chandola V, Banerjee A, Kumar V. Anomaly detection: a survey. ACM Comput Surv. 2009;41(3):1–58.
13. Binglin X, Zhanhuai L. An anomaly detection method for spacecraft using ica technology. In: International Conference on Advanced Computer Science and Electronics Information (ICACSEI 2013), pp. 50–54; 2013.
14. Tomek I, et al. An experiment with the edited nearest-nieghbor rule. IEEE Trans Syst Man Cybern. 1976;6:448–52.
15. Van Hulse J, Khoshgoftaar T. Knowledge discovery from imbalanced and noisy data. Data Knowl Eng. 2009;68(12):1513–42.
16. Jeatrakul P, Wong KW, Fung CC. Data cleaning for classification using misclassification analysis. J Adv Comput Intell Intell Inform. 2010;14(3):297–302.
17. Wilson DL. Asymptotic properties of nearest neighbor rules using edited data. IEEE Trans Syst Man Cybern. 1972;3:408–21.
18. Khoshgoftaar TM, Rebours P. Improving software quality prediction by noise filtering techniques. J Comp Sci Technol. 2007;22(3):387–96.
19. Sáez JA, Galar M, Luengo J, Herrera F. Inffc: an iterative class noise filter based on the fusion of classifiers with noise sensitivity control. Inform Fusion. 2016;27:19–32.
20. Wu Y, Liu Y. Robust truncated hinge loss support vector machines. J Am Stat Assoc. 2007;102(479):974–83.
21. Rätsch G, Schölkopf B, Smola AJ, Mika S, Onoda T, Müller K-R. Robust ensemble learning for data mining. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 341–344; 2000. Springer.
22. Zhang W, Wang D, Tan X. Data cleaning and classification in the presence of label noise with class-specific autoencoder. In: International Symposium on Neural Networks, pp. 256–264; 2018. Springer.
23. Wang D, Tan X. Bayesian neighborhood component analysis. IEEE Trans Neural Netw Learn Syst. 2017;29(7):3140–51.
24. Zhang W, Tan X. Combining outlier detection and reconstruction error minimization for label noise reduction. In: 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 1–4 ; 2019. IEEE
25. Deng L. The mnist database of handwritten digit images for machine learning research [best of the web]. IEEE Signal Process Magaz. 2012;29(6):141–2.
26. Bartholomew D. Principal components analysis. International Encyclopedia of Education, 3rd edn., pp. 374–377. New York: Elsevier; 2010.
27. Najafabadi MM, Khoshgoftaar TM, Calvert C, Kemp C. User behavior anomaly detection for application layer ddos attacks. In: 2017 IEEE International Conference on Information Reuse and Integration (IRI), pp. 154–161; 2017. IEEE.
28. Lakhina A, Crovella M, Diot C. Diagnosing network-wide traffic anomalies. ACM SIGCOMM Comp Commun Rev. 2004;34(4):219–30.
29. Najafabadi MM. Machine Learning Algorithms for the Analysis and Detection of Network Attacks. Florida Atlantic University; 2017.
30. Callegari C, Gazzarrini L, Giordano S, Pagano M, Pepe T. Improving pca-based anomaly detection by using multiple time scale analysis and kullback-leibler divergence. Int J Commun Syst. 2014;27(10):1731–51.
31. Paffenroth R, Kay K, Servi L. Robust pca for anomaly detection in cyber networks. arXiv preprint arXiv:1801.01571 2018.
32. Hyvärinen A, Oja E. A fast fixed-point algorithm for independent component analysis. Neural Comput. 1997;9(7):1483–92.
33. Hyvärinen A, Karhunen J, Oja E. What is Independent Component Analysis?, Independent Component Analysis. New York: Wiley; 2002.
34. Hyvärinen A, Oja E. Independent component analysis: algorithms and applications. Neural Netw. 2000;13(4–5):411–30.
35. Reza MS, Ruhi S. Multivariate outlier detection using independent component analysis. Sci J Appl Math Stat. 2015;3(4):171–6.
36. Chicco D, Sadowski P, Baldi P. Deep autoencoder neural networks for gene ontology annotation predictions. In: Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 533–540; 2014.
37. Najafabadi MM, Villanustre F, Khoshgoftaar TM, Seliya N, Wald R, Muharemagic E. Deep learning applications and challenges in big data analytics. J Big Data. 2015;2(1):1.
38. Almotiri J, Elleithy K, Elleithy A. Comparison of autoencoder and principal component analysis followed by neural network for e-learning using handwritten recognition. In: 2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT), pp. 1–5; 2017. IEEE.
39. Kingma DP, Welling M. An introduction to variational autoencoders. Found Trends Mach Learn. 2019;12(4):307–92.

40. Zhou C, Paffenroth RC. Anomaly detection with robust deep autoencoders. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 665–674; 2017.
41. Ranjan C. Build the Right Autoencoder – Tune and Optimize using PCA Principles. https://towardsdatascience.com/build-the-right-autoencoder-tune-and-optimize-using-pca-principles-part-i-1f01f821999b
42. Tomek I, et al. Two modifications of cnn. IEEE Trans Syst Man Cybern. 1976;11:769–72.
43. He H, Ma Y. Imbalanced Learning: Foundations, Algorithms, and Applications. New York: Wiley; 2013.
44. Brownlee J. Undersampling Algorithms for Imbalanced Classification. https://machinelearningmastery.com/undersampling-algorithms-for-imbalanced-classification/.
45. Agarwal R. The 5 Most Useful Techniques to Handle Imbalanced Datasets. https://www.kdnuggets.com/2020/01/5-most-useful-techniques-handle-imbalanced-datasets.html.
46. Kaggle: Credit Card Fraud Detection. https://www.kaggle.com/mlg-ulb/creditcardfraud.
47. Leevy JL, Khoshgoftaar TM, Bauder RA, Seliya N. A survey on addressing high-class imbalance in big data. J Big Data. 2018;5(1):42.
48. Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial intelligence-Volume 2, pp. 1137–1143; 1995. Morgan Kaufmann Publishers Inc.
49. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, et al. Scikit-learn: Machine learning in python. J Mach Learn Res. 2011;12:2825–30.
50. sklearn.decomposition.FastICA: FastICA: a fast algorithm for Independent Component Analysis. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html.
51. Gulli A, Pal S. Deep Learning with Keras. New York: Packt Publishing Ltd; 2017.
52. Seliya N, Khoshgoftaar TM, Van Hulse J. A study on the relationships of classifier performance metrics. In: Tools with Artificial Intelligence, 2009. ICTAI'09. 21st International Conference On, pp. 59–66; 2009. IEEE
53. Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Inform Sci. 2014;259:571–95.
54. Bauder RA, Khoshgoftaar TM. The effects of varying class distribution on learner behavior for medicare fraud detection with imbalanced big data. Health Inform Sci Syst. 2018;6(1):9.
55. Iversen GR, Wildt AR, Norpoth H, Norpoth HP. Analysis of variance. New York: Sage; 1987.
56. Tukey JW. Comparing individual means in the analysis of variance. Biometrics. 1949;1:99–114.

Publisher's Note