Analysis of Encoder Representations as Features Using Sparse Autoencoders in Gradient Boosting and Ensemble Tree Models

Luis Aguilar[1(✉)] and L. Antonio Aguilar[2(✉)]

[1] Academic Department of Mathematics, National University of Piura, Urb. Miraflores s/n, Castilla Apartado Postal 295, Piura, Peru
laguilari@unp.edu.pe

[2] Computer and System School, Antenor Orrego Private University, Av. América Sur 3145 Urb. Monserrate, Trujillo, Peru
laguilarg@upao.edu.pe

Abstract. The performance of learning algorithms relies on factors such as the training strategy, the parameter tuning approach, and data complexity; in this scenario, extracted features play a fundamental role. Since not all the features maintain useful information, they can add noise, thus decreasing the performance of the algorithms. To address this issue, a variety of techniques such as feature ex-traction, feature engineering and feature selection have been developed, most of which fall into the unsupervised learning category. This study explores the generation of such features, using a set of k encoder layers, which are used to produce a low dimensional feature set F. The encoder layers were trained using a two-layer depth sparse autoencoder model, where PCA was used to estimate the right number of hidden units in the first layer. Then, a set of four algorithms, which belong to the gradient boosting and ensemble families were trained using the generated features. Finally, a performance comparison, using the encoder features against the original features was made. The results show that by using the reduced features it is possible to achieve equal or better results. Also, the approach improves more with highly imbalanced data sets.

Keywords: Unsupervised learning · Sparse autoencoders · Feature generation Gradient boosting models · Ensemble models

1 Introduction

In a supervised learning setting, algorithms use a set of feature vectors x_i to predict a set of targets y_i, which can be discrete (classification) or continuous (regression). Typically xi is grouped into a design matrix $X^{(m,\ n)}$ where m represents the total observations and n the number of features. Algorithms use the learned features to predict new values $\hat{y}$ from unseen data. In this context, features play an important role in the algorithm's performance, due to the ability to increase or decrease it. Thus, learning the appropriate set of features becomes a crucial task in the majority of practical applications.

Therefore, it is necessary to apply some techniques or transformations to the original features in order to improve them. Such techniques are usually grouped into feature engineering, feature extraction and feature selection. Although certain debate exists about the formalism of feature engineering, there is no doubt of its effectiveness as observed in many machine learning competitions. An example of this is described in [1], where a set of diverse features were generated in order to improve the final model score.

Feature engineering can be defined as an iterative process, where a set of new features are generated using a set of transformations, which can be very simple, like obtaining the mean, or could involve a more complex set of calculations. Usually the quality of the obtained features is related to the practitioner's prior knowledge over the problem domain. The lack of this prior knowledge represents an issue in the feature generation process. To address this problem, different techniques have been proposed to generate features in an automatic fashion [2, 3]. In this context, algorithms are typically used to generate and evaluate the new features.

Some of those techniques involve the application of unsupervised algorithms such as autoencoders, which are a type of unsupervised neural network. They were introduced by Hinton [4] in 1986. Since then, many variants have been proposed, including sparse autoencoders [5], variational autoencoders [6] and denoising autoencoders [7]. These architectures introduce different variations to the original autoencoder model, but they are generally composed of two types of layers, namely encoder and decoder. The encoder has the function to learn a compress representation of the data. Typically, the encoder has k units such as $k < n$, where n represents the number of features in the data. This ensures that the encoder learn a compact representation of the data, similarly to what can be achieved with PCA. Meanwhile the decoder layer reconstructs the representation learned by the encoder to its original form [8].

The complexity of the autoencoders to generate features are related to factors such as the type of architecture used, the parameter tuning strategy, the preprocessing techniques and the data complexity. In this context, the encoder units are learning a certain type of fix features $A \in H$, where H represents the total feature space. Therefore, it is possible to assume that there are other feature spaces F that could also be explored as features by introducing variations to the number of units in the encoder layer. This represents the main topic of this study. But instead of using the common reconstructions from the decoder layer, the encoder layer is being used to learn and generate new features from the original data, using a different subset of encoder units. Using these new representations, a set of learning algorithms which belong to the gradient boosting and ensemble families are being trained. The results show that using this approach enables to explore other feature spaces F. Also, the new features are less dimensional than the original ones, obtaining similar or better auc scores, even in the presence of highly imbalanced data sets.

2 Main Idea and Motivation

Autoencoders are applied in diverse application domains. For example, due to their innate capability to maintain the local variability in lower dimensions, they can be used in feature embeddings [9]. In feature extraction tasks, they are capable of creating new features, which can be used as training data for classifiers [10]. Also, they are very efficient when applied to extract features from high dimensional data [11]. Another interesting application of autoencoders is related to data augmentation [12]. Finally, autoencoders can be used to automatize the feature generation process [13]. In all the applications discussed, the encoder units are capable of generating a new rich set of representations.

However, these representations correspond to a fix number of units in the encoder layer. Moreover, they are the result of a very careful and laborious tuning process, which correspond to the autoencoder architecture design. But if a different number of units were used in the encoder layer, then the learned representations would significantly change. In fact, a totally new set of features would be obtained. This makes us asking, which set of obtained features are better [14]. Using this fact, it is possible to hypothesize the existence of a more general feature space H. In this context, the encoder layer is learning an optimal subset $A \in H$.

Therefore, if variations are made to the encoder layer, then it is possible to obtain a new set of features F. However, there are diverse types of variations and criteria that can be implemented. For example, in [15], from a set of hidden layers, only the highest activations were kept. Also, multiple layers and other algorithms can be implemented to learn features as shown in [16]. Finally, it is also possible to introduce kernel techniques to control the learned representations [17]. However, in this study, variations to the number of encoder units are instead been made, resulting in different k-encoder layers.

However, in order to explore F a set of assumptions has to be made: (i) F must not be an infinite space, and (ii) there has to be a reasonable number of original n features to generate F. It is also important to pay attention to the autoencoder architecture used to explore F. For example, if a single encoder layer is used, it would be equivalent to a PCA reduction. Therefore, it is necessary that the autoencoder will be at least two layers in depth to guarantee a set of more complex features. This idea is illustrated in Fig. 1.

Also, the defined autoencoder in Fig. 1 must maintain sparsity in the learned representations. Therefore, a series of L1 and L2 regularizations, which can guarantee these criteria are been applied to the autoencoder layers.

Once the k-encoders learn F, it is possible to transform the original data using the learned representations from the k-encoders. Section 3 will provide a detailed approach describing the implementation of the main idea proposed here.

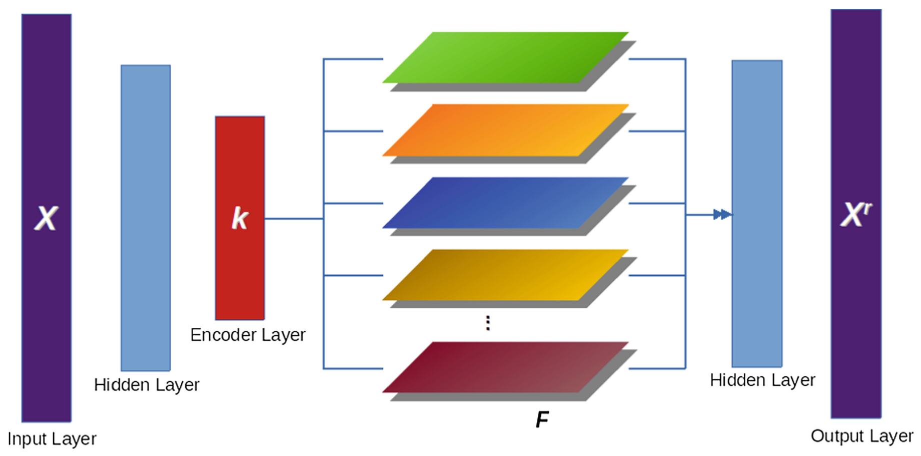

Fig. 1. Feature space F generated by the k different encoder layers.

3 Design Approach

3.1 Base Sparse Autoencoder

In order to consider the approach described in Sect. 2, a base sparse autoencoder [5] B_a will be implemented. This sparse autoencoder B_a will be used to train the different k-encoder layers. Since the architecture defined in Fig. 1 is composed by a two-layer depth sparse autoencoder, it is necessary to determine the number of units in the respective layers. Let be p the number of units in the first layer and k the number of units in the k-encoder layers. Then the number of hidden units p can be defined as follows:

$$p = \sum_{i=1}^{n} c_{var(i)} > \varepsilon \tag{1}$$

Where the total hidden units p is determined by the total principal components c_i obtained from the data set X_i, which maintain a variance greater than ϵ. In this context, ϵ becomes a parameter that must be tuned. We choose to keep $\epsilon = 0.9$. This means that the total number of units p will be equivalent to the total components that maintain a variance equal to 90%. Using (1) also acts as a filter to discard features that could represent noise or are irrelevant.

Once p is defined, it is possible to determine the total number of k-encoders units as follows:

$$k = \{1, \ldots, p - 1\} \tag{2}$$

Using (2), k is being constrained to explore a representative finite space from F. Meanwhile (1) is removing unnecessary features from X_i. To guarantee the sparsity in B_a, a set of L1 and L2 regularizations will be applied to the layers. The activation functions used are: relu for the first layer, tanh for the encoder layer, relu for the first layer of the decoder and finally sigmoid for the output layer. Also, the weights are being initialized using a normal glorot distribution (3). Finally, to guarantee the reconstruction from the original X_i, KL divergence (4) is used as a loss function.

$$stdev = \sqrt{\frac{2}{u_i + u_o}} \tag{3}$$

$$KL(q \,||\, p) = \sum_d q(d) * \log\left(\frac{q(d)}{p(d)}\right) \tag{4}$$

It is important to note that the number of k-encoders to be generated will be restricted by three factors: (i) the total number of features n in X_i, (ii) the number of units p obtained applying (1) on X_i and (iii) the number of k encoder units generated from (2). This means that per each data set X_i a set of k-encoder layers will be generated, varying only in the number of units k. Using the definition of B_a, the architecture can be summarized in Fig. 2.

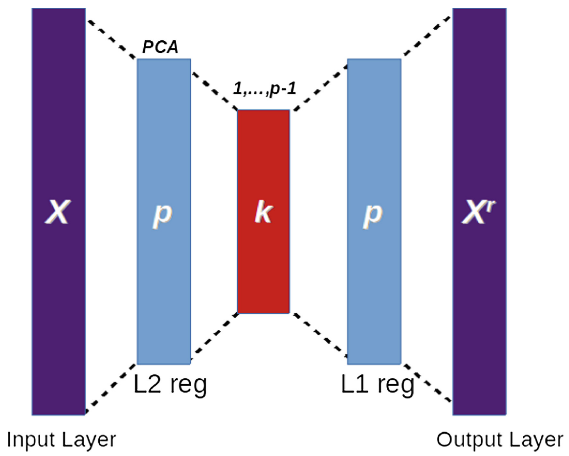

Fig. 2. Architecture representation of the B_a sparse autoencoder.

Libraries scikit-learn [18] and Keras [19] were used to implement $B_{a.}$

3.2 Data Sets

In order to test the proposed approach, library PMLB [20] was used. This library contains a well-known easily accessible benchmark data set repository. The repository can be used for regression and classification tasks. In this experiment, the data sets were restricted to only binary classification tasks. It also included both balanced and imbalanced data sets.

From the repository, a total of 13 data sets were selected. Each data set was partitioned into training (80%), development (10%) and test (10%), respectively. This partition was used during training both the sparse autoencoders B_a and the algorithms. Also, to guarantee the reproducibility of the results, a constant random seed was maintained in each split during all the experiment. Table 1 shows the data sets alongside their respectively properties.

Table 1. Data sets description.

Data Sets	# Features	# Instances	Class balance (%)	
			Positive	Negative
mushroom	22	8124	48.20	51.80
twonorm	20	7400	49.96	50.04
ring	20	7400	50.49	49.51
agaricus-lepiota	22	8145	48.09	51.91
parity5+5	10	1124	50.44	49.56
threeOf9	9	512	46.48	53.52
monk1	6	556	50.00	50.00
vote	16	435	38.62	61.38
spect	22	267	79.40	20.60
tic-tac-toe	9	958	65.34	34.66
flare	10	1066	17.07	82.93
phoneme	5	5404	29.35	70.65
xd6	9	973	33.09	66.91

3.3 Trained Algorithms

Algorithms trained with the k-encoder representations belong to the gradient boosting and ensemble families. These algorithms were LightGBM [21] and CatBoost [22], which apply gradient boosting techniques to grow trees, and Ada Boost [23] and Random Forest [24], which combine a set of week learners in order to boost their performance. To ensure that the algorithm's parameter do not influence the results, no parameter tuning was implemented. Therefore, their default configuration with a constant random seed was used during training with the k-encoder features and with the original ones.

3.4 Training Methodology

Using the definition described in (1) and (2) the right number of p units and k-encoder layers were determined for each data set X_i. Then each k-encoder layer was trained using a sparse architecture. Once the training was complete, the different k-encoders were used to transform the original features into their new feature representation F for each X_i. This process was carried out a total of ten times per each data set X_i. Then both the ensemble and gradient boosting algorithms were trained using the generated features and the original ones. Also, the auc score was averaged for the transformed test features per each data set X_i. Finally, the hardware used was an Intel core i5-6200U CPU – 2.30 GHz laptop with 8 GB RAM and running Ubuntu 16.04 LTS 64-bit.

4 Results

The experiment was performed ten times on the data sets. The table below shows the average auc test scores obtained by each algorithm, alongside with the number of p units and the best encoder layer (b_k).

The results in Table 2 indicate that each algorithm obtain their maximum auc score from different k-encoders. Also, LightGBM and CatBoost do not present much improvement from the k-encoder representation. In fact, in most cases the auc scores are equal or slightly better. However, it must be noted that these results are achieved using just a smaller number of features than the original ones, which indicates that the obtained features are relevant. On the other hand, AdaBoost and Random Forest have received a more significant improvement over their auc scores, especially Ada Boost, which achieves an average auc score greater than the normal Ada Boost model. Also, there are two remarkable cases where all the algorithms benefit greatly from the encoder representations: the spect and flare data sets.

Table 2. ROC scores from transformed test data using the best encoder layers (b_k).

Data sets	p	b_k	Ada Boost		b_k	Random forest		b_k	LightGBM		b_k	CatBoost	
			Enc	Norm		Enc	Norm		Enc	Norm		Enc	Norm
mushroom	13	12	0.97	1.00	7	**1.00**	1.00	9	**1.00**	1.00	8	**1.00**	1.00
twonorm	18	16	**0.97**	0.97	17	**0.96**	0.95	16	0.97	0.98	16	**0.98**	0.98
ring	18	17	0.90	0.96	12	0.89	0.93	16	0.92	0.97	16	0.91	0.97
agaricus-lepiota	13	11	0.95	1.00	9	**1.00**	1.00	11	**1.00**	1.00	9	**1.00**	1.00
parity5+5	9	6	**0.65**	0.43	8	**0.72**	0.57	6	0.69	1.00	6	0.68	1.00
threeOf9	9	5	**0.82**	0.81	8	0.81	0.96	5	0.84	1.00	7	0.81	1.00
monk1	6	5	**0.75**	0.70	3	0.79	0.93	5	0.88	1.00	5	0.86	1.00
vote	10	2	**1.00**	1.00	8	**1.00**	1.00	9	**1.00**	1.00	6	**1.00**	1.00
spect	14	8	**0.78**	0.53	1	**0.75**	0.68	8	**0.73**	0.58	3	**0.70**	0.68
tic-tac-toe	8	6	0.69	0.79	5	0.70	0.90	4	0.69	1.00	3	0.68	1.00
flare	7	5	**0.62**	0.55	1	**0.62**	0.59	2	**0.64**	0.59	5	**0.65**	0.59
phoneme	4	1	**0.73**	0.71	3	0.71	0.85	3	0.71	0.85	1	0.71	0.83
xd6	9	4	**0.88**	0.76	7	0.94	1.00	8	0.95	1.00	8	0.95	1.00
Total avg			***0.82***	***0.79***		***0.85***	***0.87***		***0.85***	***0.92***		***0.84***	**0.93**

When analyzing the impact of the different k-encoders over the auc performance, two cases must be noted. First, there is a point where the maximum auc score is achieved. Then, when more k-encoders units are added, the performance begins to

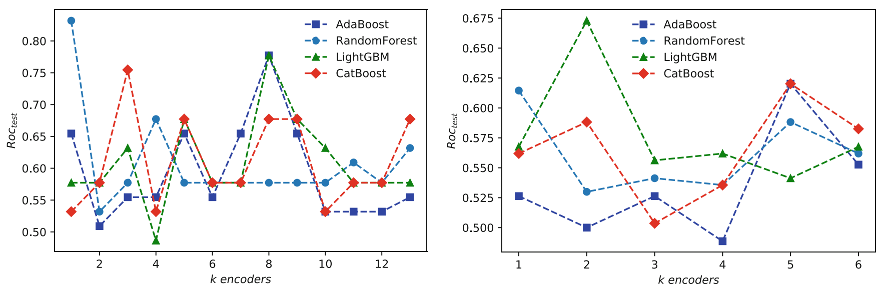

Fig. 3. Trained k-encoders on spect (left) and flare (right) data sets.

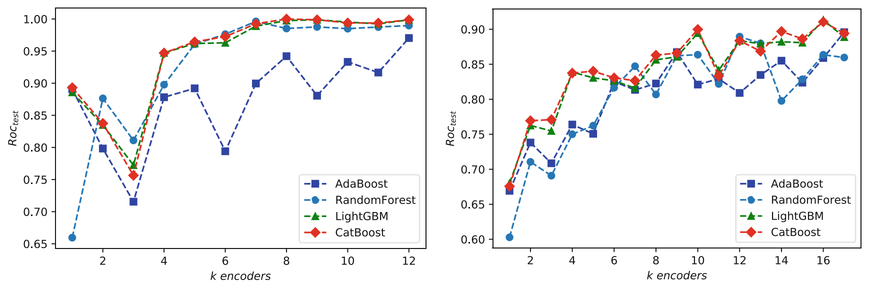

Fig. 4. Trained k-encoders on mushroom (left) and ring (right) data sets.

gradually decrease or oscillate. This pattern is more predominant with the spect and flare data sets, as observed in Fig. 3. However, this is totally the opposite with other data sets, where it seems that adding more k-encoder units improve the auc performance as seen in Fig. 4. Taking this into consideration, it is necessary to train all the k-encoders, in order to find a suitable set of features for a particular learning algorithm.

Table 3. Training time in balanced data sets.

Data sets	mushroom	twonorm	ring	agaricus-lepiota	parity5 +5	threeOf9	monk1
Total features	22	20	20	22	10	9	6
Training instances	6499	5920	5920	6516	899	409	444
K-encoders trained	12	17	17	12	8	8	5
Training time (s)	39.44	66.20	84.73	77.66	33.38	33.73	19.42

Table 4. Training time in imbalanced data sets.

Data sets	vote	spect	tic-tac-toe	flare	phoneme	xd6
Total features	16	22	9	10	5	9
Training instances	348	213	766	852	4323	778
K-encoders trained	9	13	7	6	3	8
Training time (s)	36.97	57.21	37.23	36.03	30.94	51.21

Another factor to consider is the time taken to train the k – encoders. This is illustrated in Tables 3 and 4 for the balanced and imbalanced data sets. In both cases, the k-encoders also reflect the new low dimensionally obtained features. Also, observing the results in both tables, the data sets which take more time to train the k-encoders are usually the ones that have more features. The number of training instances also have impact on the training time.

5 Conclusions

The proposed approach allows us to observe how the different k-encoder features F impacted the performance of the gradient boosting and ensemble algorithms. Also, the proposed approach improved greatly the performance in two highly imbalanced data sets, such as flare and spect. The proposed approach also allowed us to obtain better results using less dimensions than the original features.

It was also observed that in some cases, using a reduced number of k-encoders was enough to generate a good set of features F. At the same time, there were cases in which adding more k-encoders further improved the performance. Therefore, both cases are strongly related to the particular characteristics of the data sets. However, this approach should be able to generate useful features in the presence of balanced and imbalanced data sets.

When analyzing the average performance obtained by the algorithms, Ada Boost was the one which beneficiated the most of the proposed approach. In fact, a normal Ada model only obtains a 79% average auc score, in contrast with the 82% using the proposed approach.

It is also important to note that each algorithm selected a different k-encoder, which means that each k-encoder feature will have a different relevance for each algorithm. Therefore, to put this approach into practice, all the k-encoders must be trained in order to find the best representation suited for a particular algorithm.

During the experiments a limitation in the proposed approach for the k-encoder generation was found. If the number of units p for a particular data set are equal to 1, then (2) will become 0, thus, the k-encoders cannot be generated. Finally, it was observed that the addition of regularization techniques such as L1 and L2 to the layers improved significantly the results.

References

1. Martínez-Romo, J.C., Luna-rosas, F.J., Mora-gonzález, M., De Luna-ortega, C.A.: Optimal feature generation with genetic algorithms and FLDR in a restricted-vocabulary speech recognition system. In: Bio-Inspired Computational Algorithms and Their Applications, pp. 235–262 (2012). https://doi.org/10.5772/36135
2. Cheng, W., Kasneci, G., Graepel, T., Stern, D., Herbrich, R.: Automated feature generation from structured knowledge. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM 2011, p. 1395 (2011). https://doi.org/10.1145/2063576.2063779
3. Katz, G., Shin, E.C.R., Song, D.: ExploreKit: automatic feature generation and selection. In: Proceedings - IEEE 16th International Conference on Data Mining (ICDM), pp. 979–984 (2016). https://doi.org/10.1109/ICDM.2016.0123
4. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature **323**, 533–536 (1986). https://doi.org/10.1038/323533a0
5. Ng, A.: Sparse autoencoder. In: CS294A Lecture Notes, pp. 1–19 (2011). http://web.stanford.edu/class/cs294a/sae/sparseAutoencoderNotes.pdf
6. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes, pp. 1–14 (2013). https://arxiv.org/abs/1312.6114
7. Vincent, P., Larochelle, H., Bengio, Y., Manzagol, P.-A.: Extracting and composing robust features with denoising autoencoders. In: Proceedings of 25th Annual International Conference on Machine Learning, ICML 2008, pp. 1096–1103 (2008). https://doi.org/10.1145/1390156.1390294
8. Baldi, P.: Autoencoders, unsupervised learning, and deep architectures. In: Guyon, I., Dror, G., Lemaire, V., Taylor, G.W., Silver, D.L. (eds.) ICML Unsupervised and Transfer Learning, pp. 37–50 (2012). JMLR.org
9. Yu, W., Zeng, G., Luo, P., Zhuang, F., He, Q., Shi, Z.: Embedding with autoencoder regularization. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS (LNAI), vol. 8190, pp. 208–223. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40994-3_14
10. Bosch, N., Paquette, L.: Unsupervised deep autoencoders for feature extraction with educational data. In: Deep Learning with Educational Data Workshop at the 10th International Conference on Educational Data Mining (2017)
11. Meng, Q., Catchpoole, D., Skillicom, D., Kennedy, P.J.: Relational autoencoder for feature extraction. In: Proceedings of International Joint Conference Neural Networks, May 2017, pp. 364–371 (2017). https://doi.org/10.1109/ijcnn.2017.7965877
12. DeVries, T., Taylor, G.W.: Dataset augmentation in feature space, pp. 1–12 (2017). https://arxiv.org/abs/1702.05538v1
13. Yousefi-azar, M., Varadharajan, V., Hamey, L., Tupakula, U.: Autoencoder-based feature learning for cyber security applications. In: International Joint Conference on Neural Networks 2017 (IJCNN), pp. 3854–3861 (2017). https://doi.org/10.1109/IJCNN.2017.7966342
14. Bengio, Y., Courville, A., Vincent, P.: Representation learning: a review and new perspectives. IEEE Trans. Pattern Anal. Mach. Intell. **35**, 1798–1828 (2013). https://doi.org/10.1109/TPAMI.2013.50
15. Makhzani, A., Frey, B.: k-sparse autoencoders (2013). https://arxiv.org/abs/1312.5663
16. Ju, Y., Guo, J., Liu, S.: A deep learning method combined sparse autoencoder with SVM. In: 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, pp. 257–260. IEEE (2015). https://doi.org/10.1109/CyberC.2015.39

17. Kampffmeyer, M., Løkse, S., Bianchi, F.M., Jenssen, R., Livi, L.: Deep kernelized autoencoders. In: Sharma, P., Bianchi, F. (eds.) Image Analysis. SCIA 2017. LNCS, vol. 10269, pp. 419–430. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59126-1_35
18. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
19. Chollet, F.: Keras. GitHub Repos (2015). https://keras.io/
20. Olson, R.S., La Cava, W., Orzechowski, P., Urbanowicz, R.J., Moore, J.H.: PMLB: a large benchmark suite for machine learning evaluation and comparison. BioData Min. **10**, 36 (2017). https://doi.org/10.1186/s13040-017-0154-4
21. Ke, G., Meng, Q., Wang, T., Chen, W., Ma, W., Liu, T.-Y.: LightGBM: a highly efficient gradient boosting decision tree. Adv. Neural. Inf. Process. Syst. **30**, 3148–3156 (2017)
22. Dorogush, A.V., Ershov, V., Yandex, A.G.: CatBoost: gradient boosting with categorical features support. In: Workshop on ML System, NIPS 2017, pp. 1–7 (2017)
23. Hastie, T., Rosset, S., Zhu, J., Zou, H.: Multi-class AdaBoost. Stat. Interface **2**, 349–360 (2009). https://doi.org/10.4310/SII.2009.v2.n3.a8
24. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001). https://doi.org/10.1023/A:1010933404324