Записки научных
семинаров ПОМИ
Том 530, 2023 г.

D. Kushchuk, M. Ryndin

NEURON COVERAGE MAXIMIZATION FOR EFFECTIVE TEST SET CONSTRUCTION WITH RESPECT TO THE MODEL

Abstract. Real world data is not stationary and thus models must be monitored in production. One way to be sure in a model's performance is regular testing. If the labels are not available, the task of minimizing the labeling cost can be formulated. In this work, we investigate and develop various ways to construct a minimum test set for a given trained model, in a fashion where the accuracy of the model calculated on the chosen subset is as close to the real one as possible. We focus on the white box scenario and propose a novel approach that uses neuron coverage as a observable functional to maximize in order to minimize the number of samples. We evaluate the proposed approach and compare it to Bayesian methods and stratification algorithms that are the main approaches to solve this task in literature. The developed method shows approximately the same level of performance but has a number of advantages over its competitors. It is deterministic, thus eliminating the dispersion of the results. Also, this method can give one a hint on the optimal budget.

§1. Introduction

The modern world contains a lot of data, and machine learning techniques are actively used to process and analyze it. In addition, the data is constantly changing over time in response to events in the environment. Due to the so-called feature and concept drifts, model performance may degrade [15], [8]. If we use a model in production, quality loss can lead to a large material loss. Therefore, to effectively use a model one must always know how well a given model performs and adapts to changes in data.

To confirm the above, as an example, consider the results of the competition SentiRuEval-2015 [6]. During this competition, participants had to develop a machine learning model to determine if a message from Twitter has positive or negative sentiment. In 2015, data was collected from Twitter, separately for model training and separately for model testing.

Key words and phrases: minimum test dataset, neuron coverage, model monitoring.

As a result, competitors managed to develop models that show a quality of about 80% [11]. Further, in 2016, a similar competition SentiRuEval-2016 [7] was held, during which new data was collected from Twitter to train and test the model. Interestingly, existing models that were used in 2015 showed a quality of 60% on the new test data from 2016. This happened because there have been significant changes in data over this year. Over time, the political situation in the world has changed, the comments have changed their subject matter, they used various new expressions, modern jargon, words borrowed from other languages, and so on.

Note that there are many ways to get the data to test the model, but without manual labeling of the collected data it is impossible to assess the quality of the algorithm. One needs to know the correct answers that have to be obtained manually, that is, labeled. In practice, this is always a laborious process that requires time and resources. Imagine building an automatic moderator of social network comments that filters toxic messages. If one manually labels all the data for testing, then the meaning of using the model is lost, since verification will be completely performed by people, and not by the algorithm.

Thus, let us assume that there is a large unlabeled dataset and a trained machine learning model. We propose to choose a minimal subsample from this test set in such a fashion that it will be possible to estimate the performance metric of the model with the greatest accuracy.

§2. Related work

We note that the stated problem is not novel and methods to solve it have already been described in literature. For example, there are approaches that employ active Bayesian estimation [3], active learning [4], or sample stratification [1], which form the desired subsample. In some methods, the authors also check whether the dataset itself is suitable for testing the model, and can also generate new data that is ideal for testing.

However, in many previous works the number of required test samples was reduced only by a factor of 2-3, so that with the initial set size of 10000 samples it would be necessary to manually label 3-5 thousand examples, which is a lot. If we want to test our model often, we want to label at most 200-500 examples every time. Moreover, all these methods are non-deterministic, that is, a unique subset is constructed during each algorithm run, and therefore these methods are characteristic by some variance of the difference between real and predicted accuracy. It may happen that at

some specific run of a non-deterministic algorithm, the difference will be too large, and as the result users will decide to make unnecessary changes to their models.

Also, all previous works that we have considered from the literature assume that the model is a black box and only the input and output of the model are known. In practice, when checking the quality of models, one almost always has access to their inner layers and weights. That is, it is possible to analyze the values of each neuron of the model for each input element. It makes sense to assume that these values can help in choosing suitable examples for testing and methods that use information about the model can be more powerful that model-agnostic ones.

§3. Approach

3.1. Formal definitions. Consider a large test set $D = \{x_1, x_2, \ldots, x_N\}$ of size N, classifier $C(x)$, and function $a(y, C(x))$, which is equal to $\mathbb{1}\{y = C(x)\}$, where y is the real answer, which is unknown.

The ultimate goal is to determine the real accuracy of the classifier C on the entire set D:

$$\mu = \frac{1}{|D|} \sum_{i=1}^{N} a(y_i, C(x_i))$$

.

We want to construct a subset L of human-labeled elements for which $n = |L| << N$. Our task is to construct the set L in such a way that the accuracy $\hat{\mu}_L$ of the classifier C calculated on the set L is as close as possible to μ. That is, the quadratic error $Err(\hat{\mu}_L) = (\mu - \hat{\mu}_L)^2$ has to be minimal.

3.2. Idea of Neural Coverage. Safe and reliable use of deep learning systems requires an accurate interpretation of their behavior. Unfortunately, testing methods developed for traditional software systems are not well suited for deep learning systems. Similar to code coverage metrics for testing conventional software, researchers have proposed neuron coverage metrics for testing the quality of a test suite and for creating new test cases [16].

Neural coverage measures the proportion of neurons that are activated in a network. It can be used to measure the quality of a test set's performance, that is, it determines how extensively the test set covers the

internal neurons of the model [5, 14]. This can be useful for finding examples that contribute more to the loss [12]. Also, using neural coverage one can generate new test cases to increase the coverage value. This has been done, e.g., in the works [2, 10].

In this work, we intend to use the following types of neural coverage.

3.2.1. *NC.* The first type of neural coverage has been introduced in [13]. Suppose that the set $N = \{n_1, n_2, \dots\}$ comprises all neurons in the model. The input test data is represented by the set $T = \{x_1, x_2, \dots\}$, and $out(n, x)$ is a function that returns the value of neuron n for element x. Also, t is the threshold value of the neuron at which it is considered activated. Formally, the value of neural coverage is defined by the formula

$$NCov(T, t) = \frac{|\{n | \exists x \in T, out(n, x) > t\}|}{|N|}.$$

That is, if a neuron has been activated by some test example, then it is considered that the entire test sample activates this neuron.

In this work, we use the threshold $t = 0.75$.

3.2.2. *k-section coverage, boundary coverages, top-k neural coverage.* The following three types of neural coverage have been introduced in [9]. The set of neurons $N = \{n_1, n_2, \dots\}$ is also considered here. For each neuron, the values low_n and $high_n$ are calculated, which are the smallest and largest value of the neuron taken on the training data. Thus, we get the segment of possible values $[low_n; high_n]$ for every neuron.

Note that for a test element $x \in T$ the model is located in its main functional area if $\forall n \in N : out(n, x) \in [low_n; high_n]$.

(i) k-sectional neural coverage (KMNCov). For an exhaustive coverage of the main functional area, the segment $[low_n; high_n]$ is divided into k equal parts, and each such part is required to be covered by test cases.

In this work, we use $k = 1$.

However, for some neuron n, it may turn out that the value of $out(n, x)$ lies outside the interval $[low_n; high_n]$, i.e., either $out(n, x) \in (-\infty; low_n)$ or $out(n, x) \in (high_n; +\infty)$. For such examples x, the model is said to lie in the corner region if $\exists n \in N : out(n, x) \in (-\infty; low_n) \cup (high_n; +\infty)$.

(ii) Neuron boundary coverage (nbc). This coverage measures how many corner regions were covered by the test set.

(iii) Top-k neural coverage (TKNCov). For an example x and neurons n_1, n_2 in the same layer, n_1 is considered to be more active than n_2 if

$out(n_1, x) > out(n_2, x)$. For the ith layer, $top_k(x, i)$ denotes the k neurons in layer i that have the largest $out(n, x)$ for a particular input x.

Top-k neural coverage measures how many neurons were the most active at least once, that is, they were included in $top_k(x, i)$ for some x.

3.3. Subsampling neural coverage algorithm. Our goal is to build the smallest subset of the full test suite that covers all possible neurons.

We hypothesize that this subset might be ideal for testing the model.

For each of the considered neural coverage definitions, it is necessary to choose the minimum number of examples so that the value of the neural coverage on these selected examples is equal to the value of the coverage on the entire test set.

Finding the smallest subset with given properties is a hard computational task, and we approximate its solution with a greedy algorithm.

We use the following algorithm:

- calculate for each test example how many neurons it covers;
- choose an example that covers as many neurons as possible;
- choose the next example that covers the largest number of remaining uncovered neurons, and so on.

Thus, as a result we obtain a subset such that the value of neural coverage on it is equal to the value in the entire test set, but containing a much smaller number of elements.

Now combining all minimal subsets for all neural coverages, we get the minimum set for full neural coverage. The size of this set will be used as the maximum for other methods.

Note that this method is completely deterministic, which means that it eliminates the variance of the accuracy result on the constructed subset. In the methods considered previously in literature, this is not the case; in them, the variance can reach 20% on some data.

Also, the developed method itself determines the optimal number of examples for labeling, but in black box methods this is a hyperparameter.

It should be noted that if one has a fixed budget, it is possible to limit the number of examples to choose and sacrifice a part of the coverage to fit into the given budget. In the above algorithm it is necessary to stop when this threshold is reached. In this case, the constructed subset will cover the neurons of the model as much as possible, taking into account the restriction on the number of samples.

§4. Experiments

4.1. Quality metric. First of all, it is necessary to determine which assessment of the quality of the model is the reference. Usually, the full test set is very large, and the actual quality of the model can be considered to be close to the quality measured on the entire large test set. Let us call this result the reference quality, or reference accuracy.

Recall that the neural coverage method is deterministic, that is, for a given trained model this method always forms the same specific test subset. For other methods from the literature, this is not the case, they are non-deterministic and the subset can always be constructed in different ways. Moreover, in some cases, the accuracy calculated on the constructed test subset is far from the reference accuracy.

That is why we do 1000 iterations of subsampling for non-deterministic methods. For each of the 1000 constructed subsets, we compute the accuracy of the model on this subset. As a result, for such methods we have 1000 values of accuracy. For the neural coverage method, since it is deterministic and the subset is always the same, there is only one accuracy value. Now we need to somehow compare the neural coverage method with third-party methods.

The first way is to compare the absolute values of the accuracy differences between the reference value and the value calculated on the subset. Since there are 1000 accuracy values for non-deterministic methods, we need to average them. For this, we take the arithmetic mean. The method for comparing absolute values of differences is shown in Fig. 1.

Also, in order to be able to compare our method with methods from the literature, we introduce a quality metric called *proportion of bad cases*.

For a third-party non-deterministic method, this metric shows the percentage of iterations out of a thousand in which the accuracy value on the constructed subset is further from the reference than the accuracy value calculated on the subset constructed by the neural coverage method.

That is, suppose we trained the model and the accuracy of the model on the test subset constructed by the neural coverage method is 91%, while the reference accuracy on full test set is 90% (the difference in accuracy values is 1%). Then, for the third-party method all accuracy values on constructed subsets (out of a thousand) that do not fall into the interval (89, 91) are considered to be further from the reference accuracy than neural coverage method. For example, in 600 cases out of 1000, the accuracy values do not fall within the interval (89, 91), but in 400 cases they do. Then the

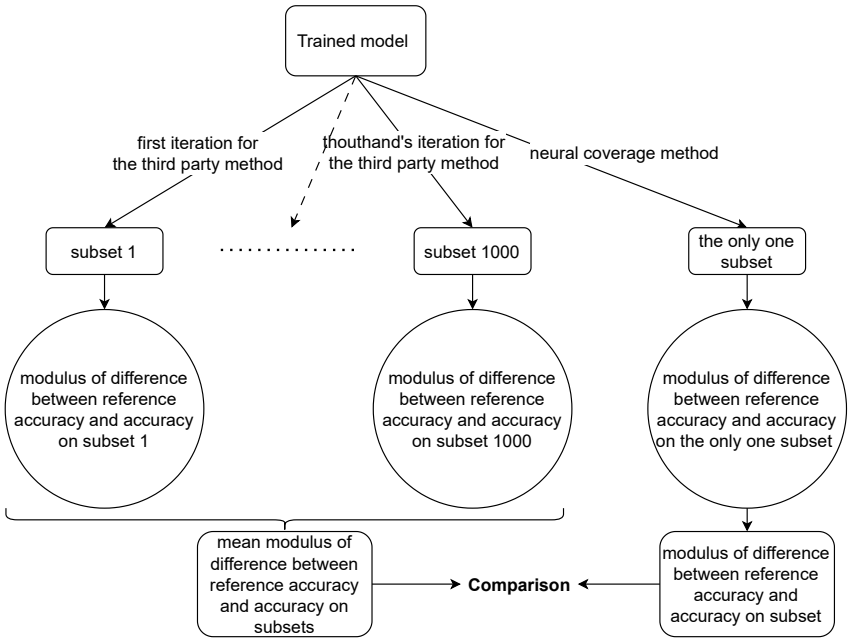

Figure 1. Comparison of the difference in absolute values of the accuracy for a third-party method and the neural coverage method.

proportion of bad cases for this third-party method is 60%. The higher the value of the metric, the worse the third-party method is relative to the neural coverage method.

The calculation of this metric is shown in detail in Fig. 2.

4.2. Experiment steps. First, we fix the dataset and the architecture of the model for it. Section 4.3 shows tables with the architecture of the models for each dataset.

Next, the model is trained on the training data. As a result, we have a trained model and we want to compare the neural coverage method with third-party non-deterministic methods.

Since third-party algorithms for test subsampling are non-deterministic, we run 1000 subsampling iterations for such methods. On every iteration,

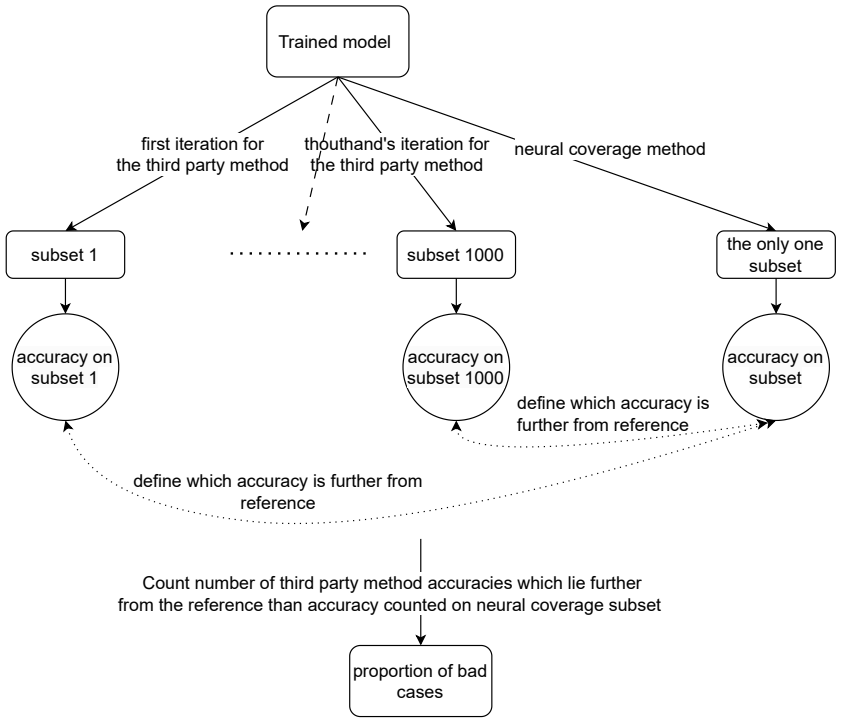

Figure 2. Calculation of the *proportion of bad cases* metric for one selected third-party method and trained model.

we get a subset and calculate the accuracy of the model on it. For each of the methods, the absolute value of the accuracy difference and the *proportion of bad cases* metric from Section 4.1 are calculated.

Note that during model training, the weights are initialized randomly. This means that the final weights of the trained model may differ. The neural coverage method directly uses the weights of the model, and therefore, for each new initialization of the weights, the neural coverage method can produce different subsets. To take this into account, for each model architecture, 10 training runs are carried out with different initial weights.

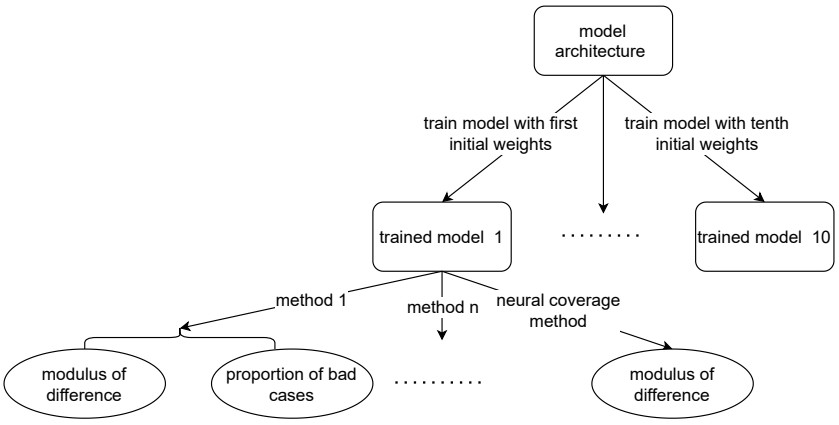

Figure 3. 10 model training runs with different initial weights and metrics calculation.

Remembering the previously introduced metrics and ten different trained models for one dataset, for each trained model we conduct experiments with each sampling algorithm and obtain the value of the mentioned metrics. This is shown in Fig. 3.

As a result, for each dataset we create 2 tables. The first contains information about the model: layers and hyperparameters.

The second table shows the arithmetic mean value of the absolute value of the difference of the accuracy from the reference among all 10 trained models, as well as the required number of examples for labeling.

For each third-party method, a vector of 10 metric values of the *proportion of bad cases* is obtained (for 10 different model training runs). The results are shown on the plotted graphs. For each method, the graph displays an 80% confidence interval and marks the median.

4.3. Experimental implementation. Experiments were carried out on the MNIST10, CIFAR10 and 20 Newsgroups datasets with the respective models shown in Tables 1, 3, and 5.

Tables 2, 4, and 6 show the average absolute values of the difference of accuracy from the reference for the model for different datasets.

Figures 4, 5, and 6 show the proportion of bad cases for each model.

Table 1. Layers and parameters of the LeNet5 model for the MNIST10 dataset

epochs = 2, optimizer = adam, loss = categorical_crossentropy		
Layer (type)	Output Shape	Param #
InputLayer	[(None, 28, 28, 1)]	0
Conv2D	(None, 24, 24, 6)	156
MaxPooling2D	(None, 12, 12, 6)	0
Conv2D	(None, 8, 8, 16)	2416
MaxPooling2D	(None, 4, 4, 16)	0
Flatten	(None, 256)	0
Dense	(None, 120)	30840
Dense	(None, 84)	10164
Dense	(None, 10)	850
Activation	(None, 10)	0
Total params: 44,426		
Trainable params: 44,426		
Non-trainable params: 0		

Table 2. Average absolute value of the difference of accuracy from the reference for the LeNet5 model for the MNIST10 dataset

Method	*Avg # of examples*	*Difference of accuracy, %*
Full test set	10000	0
Neuron coverage	122	1.23
IPrior	122	0.86
IPrior+TS	122	0.89
Perc, pps	122	**0.84**
Opt conf online	122	1.99
Random choosing	122	0.87

§5. Concept drift experiment

In these experiments, we introduce artificial concept drift, using the Amazon review dataset for that. This dataset consists of positive and negative user reviews for various types of products. We want to train a model that will distinguish between a positive comment and a negative one.

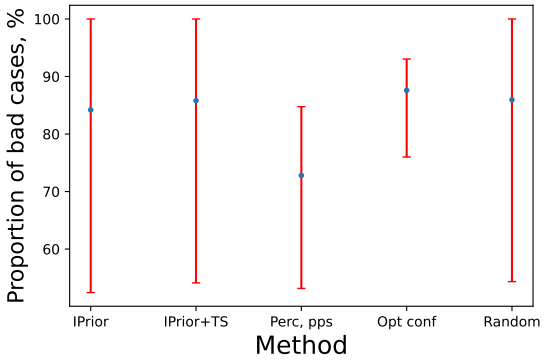

Figure 4. Proportion of bad cases for the LeNet5 model for the MNIST10 dataset

Table 3. Layers and parameters of the 3VGG model for the CIFAR10 dataset

epochs = 10, optimizer = SGD, loss = categorical_crossentropy

Layer (type)	Output Shape	Param #
Conv2D	(None, 32, 32, 32)	896
Conv2D	(None, 32, 32, 32)	9248
MaxPooling2D	(None, 16, 16, 32)	0
Conv2D	(None, 16, 16, 64)	18496
Conv2D	(None, 16, 16, 64)	36928
MaxPooling2D	(None, 8, 8, 64)	0
Conv2D	(None, 8, 8, 128)	73856
Conv2D	(None, 8, 8, 128)	147584
MaxPooling2D	(None, 4, 4, 128)	0
Flatten	(None, 2048)	0
Dense	(None, 128)	262272
Dense	(None, 10)	1290
Total params: 550,570		
Trainable params: 550,570		
Non-trainable params: 0		

Table 4. Average absolute value of the difference of accuracy from the reference for the 3VGG model for the CIFAR10 dataset

Method	*Avg # of examples*	*Difference of accuracy, %*
Full test set	10000	0
Neuron coverage	202	**2.5**
IPrior	202	2.73
IPrior+TS	202	2.63
Perc, pps	202	2.67
Opt conf online	202	4.57
Random choosing	202	2.78

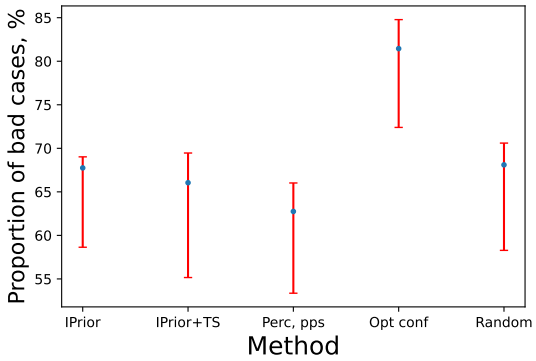

Figure 5. Proportion of bad cases for the 3VGG model for the CIFAR10 dataset

First, a model with an LSTM layer is trained on one of the types of comments. Then it is being tested on a different type of comments, thus implementing an artificial replacement of the subject. For this experiment, two types of reviews were used: about clothes and about cosmetics.

The model was trained on comments about clothes, but tested on comments about cosmetics.

Table 5. Layers and parameters of the model for the 20 Newsgroups dataset

epochs = 10, optimizer = rmsprop, loss = categorical_crossentropy		
Layer (type)	Output Shape	Param #
GloVe Embedding	(None, 1000, 100)	1000000
LSTM	(None, 128)	117248
Dense	(None, 128)	16512
Dense	(None, 20)	2580
Total params: 1,136,340		
Trainable params: 136,340		
Non-trainable params: 1,000,000		

Table 6. Average absolute value of the difference of accuracy from the reference for the model for the 20 Newsgroups dataset

Method	*Avg # of examples*	*Difference of accuracy, %*
Full test set	4000	0
Neuron coverage	177	2.88
IPrior	177	**2.63**
IPrior+TS	177	3.69
Perc, pps	177	2.69
Opt conf online	177	17.0
Random choosing	177	2.65

§6. Conclusion

In this work, we propose a novel approach to sampling a test subset from a large unlabeled set. This method is based on the idea of model neural coverage. The algorithm selects examples from a large unlabeled set that cover as many neurons as possible.

For experiments, several problems from different fields were considered. In the field of natural language processing and image processing, the neural coverage algorithm showed relatively good results.

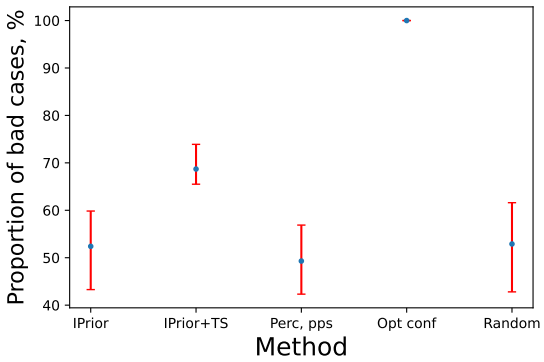

Figure 6. Proportion of bad cases for the model for the 20 Newsgroups dataset

Table 7. Layers and parameters of the model for the Amazon review dataset

epochs = 5, optimizer = rmsprop, loss = binary_crossentropy		
Layer (type)	Output Shape	Param #
InputLayer	[(None, 32)]	0
Embedding	(None, 32, 64)	768000
GRU	(None, 32, 128)	74496
GRU	(None, 128)	99072
Dense	(None, 32)	4128
Dense	(None, 100)	3300
Dense	(None, 1)	101
Total params: 949,097 Trainable params: 949,097 Non-trainable params: 0		

In experiments with text sentiment detection with artificial domain shift, the neural coverage algorithm also shows good results and outperforms third-party methods in more than 60% of cases. Thus, these experiments confirm that this algorithm can potentially be used in real problems with natural concept drift.

Table 8. Average absolute value of the difference of accuracy from the reference for the model for the Amazon review dataset

Method	*Avg # of examples*	*Difference of accuracy, %*
Full test set	10000	0
Neuron coverage	178	**2.19**
IPrior	178	3.03
IPrior+TS	178	3.03
Perc, pps	178	3.0
Opt conf online	178	2.97
Random choosing	178	3.02

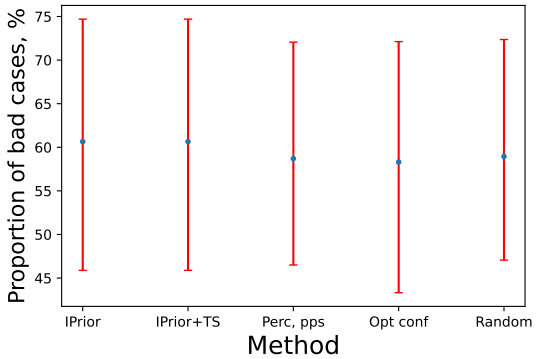

Figure 7. Proportion of bad cases for the model for the Amazon review dataset

Acknowledgements

This research was done as a part of scientific program of the National Center for Physics and Mathematics, direction #9 “Artificial intelligence and big data in technical, industrial, natural and social systems”.

References

1. P. N. Bennett and V. R. Carvalho, *Online stratified sampling: evaluating classifiers at web-scale*, Proceedings of the 19th ACM international conference on Information and knowledge management, 2010, pp. 1581–1584.
2. F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, *Is neuron coverage a meaningful measure for testing deep neural networks?*, Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 851–862.
3. D. Ji, R. L. Logan IV, P. Smyth, and M. Steyvers, *Active Bayesian assessment for black-box classifiers*, arXiv preprint arXiv:2002.06532 (2020).
4. J. Kossen, S. Farquhar, Y. Gal, and T. Rainforth, *Active testing: Sample-efficient model evaluation*, International Conference on Machine Learning, PMLR, 2021, pp. 5753–5763.
5. E. Lanus, L. J. Freeman, D. R. Kuhn, and R. N. Kacker, *Combinatorial testing metrics for machine learning*, 2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), IEEE, 2021, pp. 81–84.
6. N. Loukachevitch, P. Blinov, E. Kotelnikov, Y. Rubtsova, V. Ivanov, and E. Tutubalina, *SentiRuEval: Testing object-oriented sentiment analysis systems in russian*, Proceedings of International Conference Dialog, vol. 2, 2015, pp. 3–13.
7. N. Loukachevitch and Y. V. Rubtsova, *SentiRuEval-2016: Overcoming time gap and data sparsity in tweet sentiment analysis*, Computational Linguistics and Intellectual Technologies, 2016, pp. 416–426.
8. J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, *Learning under concept drift: A review*, IEEE Transactions on Knowledge and Data Engineering **31** (2018), no. 12, 2346–2363.
9. L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, et al., *DeepGauge: Multi-granularity testing criteria for deep learning systems*, Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018, pp. 120–131.
10. S. Mani, A. Sankaran, S. Tamilselvam, and A. Sethi, *Coverage testing of deep learning models using dataset characterization*, arXiv preprint arXiv:1911.07309 (2019).
11. V. Mayorov, I. Andrianov, N. Astrakhantsev, V. Avanesov, I. Kozlov, and D. Turdakov, *A high precision method for aspect extraction in Russian*, Komp'juternaja Lingvistika i Intellektual'nye Tehnologii, 2015, pp. 34–43.
12. A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, *TensorFuzz: Debugging neural networks with coverage-guided fuzzing*, International Conference on Machine Learning, PMLR, 2019, pp. 4901–4911.
13. K. Pei, Y. Cao, J. Yang, and S. Jana, *DeepXplore: Automated whitebox testing of deep learning systems*, proceedings of the 26th Symposium on Operating Systems Principles, 2017, pp. 1–18.
14. Y. Tian, K. Pei, S. Jana, and B. Ray, *Deeptest: Automated testing of deep-neural-network-driven autonomous cars*, Proceedings of the 40th international conference on software engineering, 2018, pp. 303–314.

15. A. Tsymbal, *The problem of concept drift: Definitions and related work*, Computer Science Department, Trinity College Dublin **106** (2004), no. 2, 58.
16. Z. Yang, J. Shi, M. H. Asyrofi, and D. Lo, *Revisiting neuron coverage metrics and quality of deep neural networks*, CoRR **abs/2201.00191** (2022).

Ivannikov Institute
for System Programming of the RAS
E-mail: dkuschu@ispras.ru

E-mail: mxrynd@ispras.ru

Поступило 6 сентября 2023 г.