SAND2022-7082C

Managing Randomness to Enable Reproducible Machine Learning

Hana Ahmed
Sandia National Laboratories
Albuquerque, New Mexico
hahmed@sandia.gov

Jay Lofstead
Sandia National Laboratories
Albuquerque, New Mexico
gflofst@sandia.gov

ABSTRACT

The National Information Standards Organization defines scientific *reproducibility* as "obtaining consistent results using the same input data, computational steps, methods, and code, and conditions of analysis" [12]. Reproducibility in machine learning (ML) refers to the ability to regenerate an ML model precisely guaranteeing identical accuracy and transparency. While a model may offer reproducible inference, reproducing the model itself is frequently problematic at best due to the presence of pseudo-random numbers as part of the model generation. One way to ensure that models are trustworthy is by managing the random numbers produced during model training. This paper establishes examples of the impact of randomness in model generation and offers a preliminary investigation into how random number generation can be controlled to make ML models more reproducible.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning algorithms**.

KEYWORDS

machine learning, reproducibility, randomness, pseudo-random number generator, neural network

ACM Reference Format:
Hana Ahmed and Jay Lofstead. 2022. Managing Randomness to Enable Reproducible Machine Learning. In *Proceedings of the 5th International Workshop on Practical Reproducible Evaluation of Computer Systems (P-RECS '22), June 30, 2022, Minneapolis, MN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526062.3536353

1 BACKGROUND

Machine learning methods are applied to a wide range of fields and modern problems. ML has been pivotal to advancements in fields such as health care [4, 5], material sciences [7], economics [1], radiology [8], civil engineering [19], and sales and marketing [23]. ML algorithms have been used to solve recommendation systems [6, 21], image recognition [11], email spam detection [24], machine translation [14], desktop assistance[1], and sentiment analysis [9, 17].

In ML, as in most scientific fields, experimental reproducibility is a benchmark for verifying scientific findings. The majority of scientists will fail to reproduce findings of a prior study [2], informing what is considered a "reproducibility crisis" across scientific fields. Many scientific findings are in fact the results of repeated retrial and retesting until the desired results are achieved, without emphasizing that the results are actually extremely rare and unlikely to obtain by running the same experiment [13].

However, the reproducibility of random number sequences in ML models is often overlooked. Randomness is a tool used in ML to train more robust and accurate models. What is randomized during the training process (i.e., training data, input features, and initial weights) will vary across ML algorithms. Random numbers are generated in sequences by *pseudo-random number generators* (PRNGs). When different number sequences are produced, the resulting ML model will also be different. Thus, being able to reproduce an ML model's random number sequence is critical to its reproducibility—and therefore its scientific trustworthiness.

In recent decades, computer scientists have regenerated pseudo-random number sequences by recording the PRNG *seed* (the starting value of a pseudo-random number sequence). Sen et al. [22] use a "capture-and-replay" method to replay data races using the same PRNG seed. Frederickson et al. [10] uses a similar approach to reproduce Monte Carlo trees.

This paper explores the variance in model performance that results from different sequences of random numbers being produced in the training process. In this study, random number generation is controlled by regulating the pseudo-random number generator seed to ensure the same generation process is used each time. We designed C++ intercepts to `std::rand()` and `std::srand()` and store any seed used either deliberately by the caller or generated by the intercept. By using this approach, simply adding a new include file line and linking in the replacement code can alter the random number generation and add reproducibility features without having to modify the ML algorithm source itself. This minimally invasive approach we think is key for widespread adoption of reproducibility for ML algorithms and demonstrate the effectiveness of this approach. This intercept was used in a series of experiments with various ML algorithms to explore the relationship between random number generation and model accuracy on public, commonly used data sets.

Randomness in neural networks and the importance has been investigated by Scardapane, et. al [20]. The impacts of training data, specifically over-sampling and under-sampling methods, on model

[1] http://www.ai.sri.com/project/CALO

P-RECS '22, June 30, 2022, Minneapolis, MN, USA

ACM ISBN 978-1-4503-9313-3/22/06…$15.00
https://doi.org/10.1145/3526062.3536353

quality are observed by Batista et. al [3]. This motivates our study into the results of varying randomness and training/testing data in the development of ML models.

Zhang et. al [25] investigate why deep neural network (DNN) models successfully generalize and avoid overfitting to training data. This seems to avoid the need to address randomness for DNNs. Conversely, Raghunathan et. al [18] find that training with adversarial examples can deter generalization in neural networks. While each of these works discuss the impact of randomness indirectly, they do not address the root problem randomness introduces into the studies making general conclusions difficult to trust. In short, DNNs may be less affected, but only if the training data is friendly enough. This is impossible to predict for a very large data set, such as those frequently used for DNNs.

Due to the black box nature of ML models, it is difficult to determine how randomized aspects such as training data, input features, and initial weights influence the final predictive quality. Our understanding as both programmers and users of how a prediction was made and how a model might interact with real data becomes limited. This creates a need for advanced interpretability techniques to deconstruct the black box, as addressed by Krause, et. al [15]. To properly test varying these parameters, the randomness inherent in the underlying algorithms must also be addressed or it is not a fair comparison as our results demonstrate. Our paper reveals that controlling generated random number sequences is a necessary step to reproducing a given ML model.

2 DESIGN

The software design for these studies is to intercept calls to the built-in PRNG and offer both a better replacement as well as recording the crucial seed that enables re-creating the model exactly by a completely different research group.

The C++ Standard Library possesses multiple PRNGs. The original holdover from C, `std::rand()` and `std::srand()` remain. `std::rand()` is considered to be an inefficient PRNG, as it is a slower algorithm, has a limited range of $[0, 32767]$[2], uses a linear congruential engine (a comparatively low quality approach with a short period)[3,4], and produces non-uniform results. Alternatively, the C++11 `<random>` library contains better quality PRNGs.

We developed replacement code which, when included in a C++ build, intercepts any given call to `std::rand()` or `std::srand()`. The former calls an mt19937 generator (a Mersenne twister [16]). The latter records the seed for replay. The algorithms are illustrated in Algorithms 1 and 2. When `std::srand()` is not called, a random seed is generated using a `time()` function. Then, by intercepting `std::srand()` at its call, we are able to store the seed, which determines the sequence of numbers that will be output by the generator, for later reference. This simple approach is sufficient to intercept and store the start of the pseudo-random number sequence enabling replay. More advanced techniques, such as recording all of the numbers generated from a hardware random number generator, are straightforward extensions of this approach.

We validated our approach by considering four use cases:

(1) `std::srand()` has been called to set a seed for the first time in the program.
(2) `std::srand()` has been called to set a seed after it has already been called by a previous application run.
(3) `std::srand()` has not been called to set a seed, and `std::rand()` has been called for the first time.
(4) `std::srand()` has not been called to set a seed, and `std::rand()` has already been called at least once before.

The basic idea is to generate a seed if none is provided, but to always record the seed if none was saved previously. On subsequent runs, rather than generating a new seed (or using a provided one if the code is attempting to provide a new seed each run), the value from the previous run is read and used. This ensures the pseudo-random number sequence will be identical for subsequent runs as we are controlling both the PRNG algorithm and the seed value.

Algorithm 1 srand

Let *seed* be an integer
Let *seedFile* be a file
if *seedFile* **is** empty **or** !exists **then**
    Create a file named *seedFile*
    seedFile ← *seed*
else
    seed ← value in *seedFile*
end if
Seed a PRNG with *seed*

Algorithm 2 rand

if Algorithm 1 has not been run **then**
    Let *newSeed* be an integer
    newSeed ← current time
    Run Algorithm 1 where *seed* = *newSeed*
end if
Generate an integer using the seeded PRNG

One challenge is present when trying to use a mt13997 generator behind the `std::rand()` interface. For the former, the generated number is an unsigned 32-bit integer. For the latter, it is a signed 32-bit integer. To avoid dealing with negative numbers as they are completely unexpected in any code using `std::rand()`, we right bit shift by 1 the mt13997 generated number to shift it back into scope. We also redefine the `RAND_MAX` accordingly enabling algorithms that attempt to proportion data based on the function and the maximum value to continue to work properly. Ideally, the full range would be used, but that is not an option when replacing the implementation of `std::rand()`. While 32-bit precision is preferred, the period of 13997 values does not require the entire range.

All intercept code, machine learning algorithms, intercept code tests, and experimental data will later be published on Github upon paper acceptance.

3 EXPERIMENTS

Our experiments are conducted on an Intel®Core™ i7-8665U CPU @ 1.90GHz 2.11GHz, with 32 GB RAM. The operating system is

[2] https://www.cplusplus.com/reference/cstdlib/rand/
[3] https://www.cplusplus.com/reference/random/linear_congruential_engine/
[4] https://channel9.msdn.com/Events/GoingNative/2013/rand-Considered-Harmful

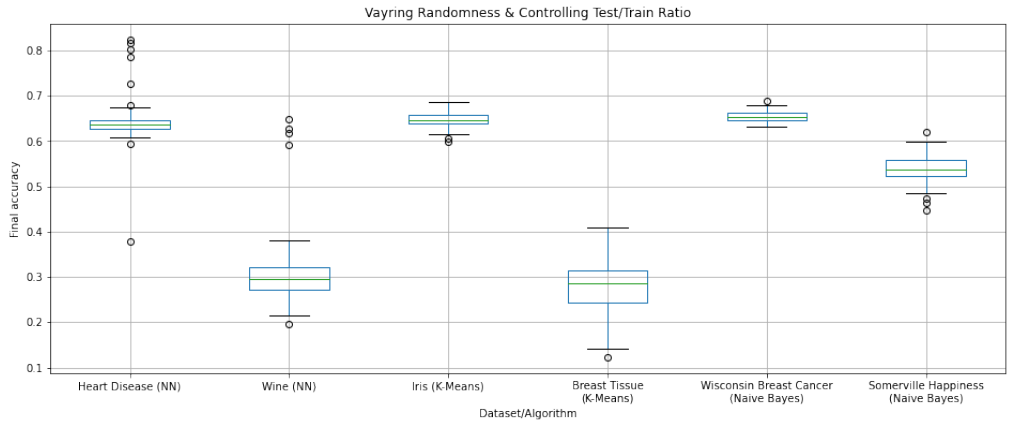

Figure 1: Varying the random seed and randomizing train/test data at a 30/70 split, the performances of 100 Neural Networks (NN) models on Heart Disease and Wine data sets, 100 K-Means clustering models on Iris and Breast Tissue, and 100 Naive Bayes models on Wisconsin Breast Cancer and Somerville Happiness.

Windows 10 and uses the C++ Clang Compiler for Windows (11.0.0) and C++ Clang-cl for v142 build tools (x64/x86). The mt13997 parameters are the C++11 standard values. Further experimentation using different numerical distributions is left for future work.

For each experiment, we train and test up to 100 models on each two different data sets for each of three algorithms for a total of six sets. The selected algorithms are neural network (NN), k-means (knnn) clustering, and naive Bayes (nB) classifier models. The six different data sets are from the UCI Machine Learning Repository and their associated algorithm are as follows: Heart Disease (NN), Wine (NN), Iris (knnn), Breast Tissue (knnn), Wisconsin Breast Cancer (nB), and Somerville Happiness (nB).

In these experiments, we focus on controlling or varying each model's 1) random seed, 2) train/test ratio, 3) training data set, and 4) testing data set. From the 16 possible permutations of these four variables, this paper covers experiments with eight of the permutations. We did not pursue the four permutations where both the seed and train/test ratio are fixed, as this combination would guarantee that the exact model is reproduced each time, making the experiments fruitless. Similarly, we did not pursue the other four permutations where both the seed and train/test ratio are varied, as this would guarantee the generation of completely different models each time, again producing no novel results or insights.

3.1 Experiment 1

In this experiment, we use `time()` as the random seed for each model, fix the train/test ratio at 30/70, and randomize the data before splitting into training and testing sets. We generate 100 models on each of the two data sets for each algorithm making for 600 models total. Results are in Figure 1 and Table 1.

First we examine the neural network models. On the Heart Disease data set, the neural networks perform from 0.3783 to 0.8237 accuracy (a 44.53% range). On the Wine data set, accuracy ranges from 0.1958 to 0.6475 (a 45.17% difference between the worst and best models). Seeing that the random number seeds were varied throughout this experiment, and no model generated had equal performance, these results demonstrate a significant impact of generated random numbers on an ML model's ultimate quality and performance.

Next, the k-means clustering models. On the Iris data set, the k-means models perform within a range of 0.5980 accuracy to 0.6868 accuracy (a 8.883% range). On the Breast Tissue data set, performance ranges from 0.1216 to 0.4078 accuracy (a 28.62% range).

The naive Bayes classifiers perform with accuracies ranging from 0.6326 to 0.689 on Wisconsin Breast Cancer (a 5.6423% range) and from 0.4466 to 0.6195 on Somerville Happiness (a 17.29% range).

3.2 Experiment 2

In this next experiment, we vary the random seed by outputting `time()` for each model, fixed the train/test ratio at 30/70, randomized the train data set, and fix the test data set. This means that for each algorithm/data set pair, the data set is shuffled before being assigned to training data. The models are then tested on the same, arbitrary test data taken from the unshuffled data set. We generate 100 models on each of the two data sets for each algorithm, making for 600 models total. Results for this experiment can be seen in Figure 2, with an additional summary in Table 2.

The neural network models on the Heart Disease data set range from 0.4873 to 0.8196 accuracy (a 33.23% difference). On the Wine data set, however, the accuracy is consistent at 0.329609.

K-means clustering models perform within 0.4646 and 0.6868 accuracy (a 22.22% range in model quality). We see a much wider variation of k-means models produced on the Breast Tissue data set, performing between 0.1973 and 0.4605 accuracy (a 26.31% difference).

Lastly, the naive Bayes algorithm produced 100 identical models for the Wisconsin Breast Cancer data set which perform at 0.65762 accuracy, and another 100 identical models for Somerville Happiness which perform at 0.6196 accuracy.

3.3 Experiment 3

In this experiment, we fix the random seed, vary the train/test ratio, vary the training data set, and fix the test data set. This means that

Table 1: Model accuracy results from varying the random seed and randomizing train/test data at a 30/70 split for each model, a summary of the final performances for 100 neural networks on Heart Disease and Wine data sets, 100 k-means models on Iris and Breast Tissue, and 100 naive Bayes models on Wisconsin Breast Cancer and Somerville Happiness.

Data set	Size	Train/test	Min	Max	Mean	Median
Heart disease (NN)	303	30/70	0.3784	0.8238	0.6411	0.6367
Wine (NN)	178	30/70	0.1958	0.6475	0.3039	0.2948
Iris (K-means)	150	30/70	0.598	0.6869	0.6472	0.6455
Breast tissue (K-means)	106	30/70	0.1216	0.4079	0.2765	0.2857
Wisconsin Breast Cancer (Naive Bayes)	699	30/70	0.6327	0.6891	0.6543	0.6535
Somerville Happiness (Naive Bayes)	143	30/70	0.4466	0.6196	0.5397	0.536

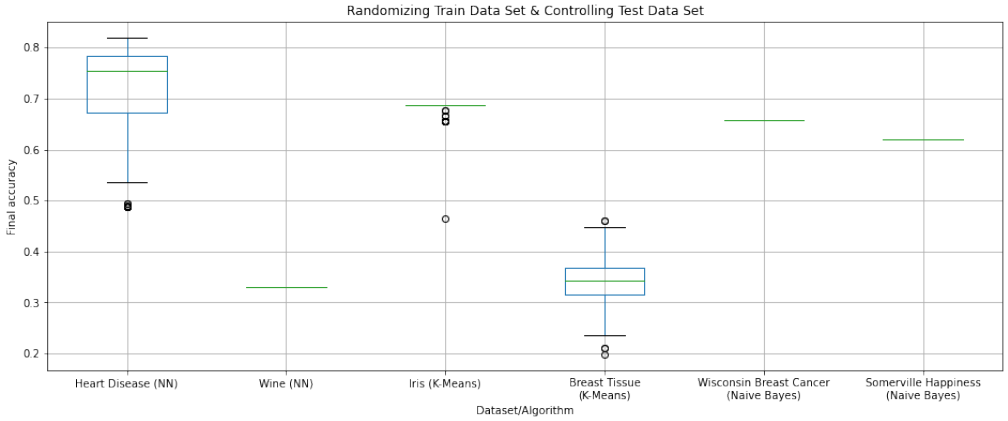

Figure 2: Varying the random seed, controlling the train/test ratio at 30/70, varying the train data set, and controlling the test data set. The figure below shows the performances of 100 Neural Networks (NN) models on Heart Disease and Wine data sets, 100 K-Means clustering models on Iris and Breast Tissue, and 100 Naive Bayes models on Wisconsin Breast Cancer and Somerville Happiness.

Table 2: Model accuracy results from fixing the test data set for each algorithm/data set pair, the performance of 100 neural network models on Heart Disease and Wine, 100 k-means models on Iris and Breast Tissue, and 100 naive Bayes models on Wisconsin Breast Cancer and Somerville Happiness.

Data set	Size	Train/Test	Min	Max	Mean	Median
Heart Disease (NN)	303	30/70	0.4873	0.8197	0.7089	0.7544
Wine (NN)	178	30/70	0.3296	0.3296	0.3296	0.3296
Iris (K-Means)	150	30/70	0.4646	0.6869	0.6819	0.6869
Breast Tissue (K-Means)	106	30/70	0.1974	0.4605	0.3389	0.3421
Wisconsin Breast Cancer (Naive Bayes)	699	30/70	0.6576	0.6576	0.6576	0.6576
Somerville Happiness (Naive Bayes)	143	30/70	0.6196	0.6196	0.6196	0.6196

for each algorithm/data set pair, we use the same random seed to produce a randomized test data set. The models are then tested on the same, arbitrary test data taken from the unshuffled data set. We generate 10 models for each algorithm, with train/test ratios at 10/90, 20/80, 30/70, 40/60, 50/50, 60/40, 70/30, 80/20, and 90/10. A summary of the results for this experiment can be seen in Table 3.

We see a variety of results in the performances of each algorithm and the models they generate. Starting with the neural network algorithm, the worst neural network on Heart Disease performs with an accuracy of 0.487329 at the train/test ratios 10/90, 20/80, 40/60, 50/50, and 90/10. The best neural network performs with 0.8372 accuracy at 80/20 (a 34.99% difference). On the Wine dataset, the worst neural network performs with an accuracy of 0 at a 10/90 and 20/80 train/test ratio, whereas the best neural network performs with 0.8333 accuracy at all other train/test ratios (an 83.33% difference).

Moving on to k-means clustering, the worst model performs on the Iris data set with an accuracy of 0 at a 90/10 train/test ratio, whereas the best k-means model performs with an accuracy of 1 at 70/30 and 80/20 (a 100% difference). On the Breast Tissue data set, the worst k-means model performs with 0.0566 accuracy at a 50/50 train/test ratio, and the best k-means model performs with 0.5454 accuracy at 90/10 (a 48.88% difference).

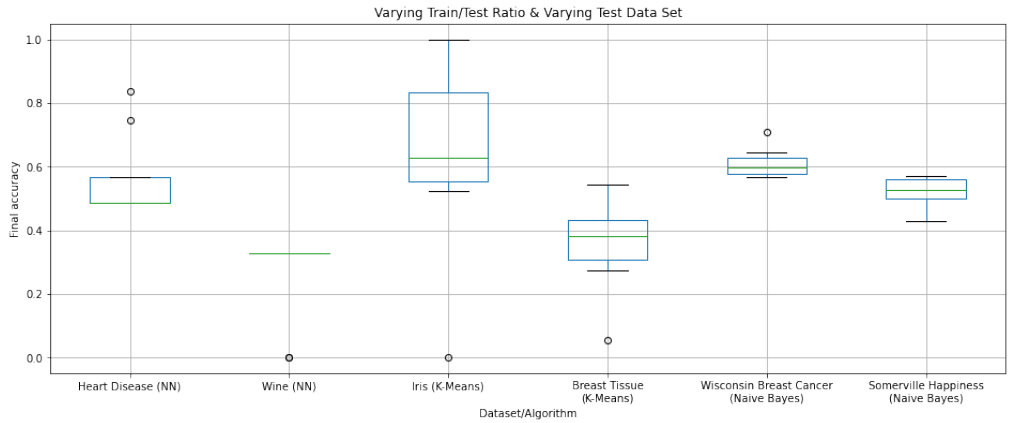

Figure 3: Controlling the random seed, varying the train/test ratio, varying the train data set, and controlling the test data set. The figure below shows the performances of 10 Neural Networks (NN) models on Heart Disease and Wine data sets, 10 K-Means clustering models on Iris and Breast Tissue, and 10 Naive Bayes models on Wisconsin Breast Cancer and Somerville Happiness.

Table 3: Fixing the random seed, varying the train/test ratio, varying the train data set, and fixing the test data set for each algorithm/data set pair. The performance of 10 neural network models each on Heart Disease and Wine, 10 k-means models each on Iris and Breast Tissue, and 10 naive Bayes models each on Wisconsin Breast Cancer and Somerville Happiness.

Train/Test	Heart Disease (NN)	Wine (NN)	Iris (K-Means)	Breast Tissue (K-Means)	Wisconsin Breast Cancer (Naive Bayes)	Somerville Happiness (Naive Bayes)
10/90	0.4873	0	0.6296	0.2737	0.6455	0.5703
20/80	0.4873	0	0.5833	0.3095	0.6279	0.5701
30/70	0.7476	0.3296	0.523	0.4324	0.6155	0.56
40/60	0.4873	0.3296	0.5556	0.4127	0.5776	0.5294
50/50	0.4873	0.3296	0.6667	0.0566	0.5673	0.5211
60/40	0.5663	0.3296	0.8333	0.3571	0.5878	0.5263
70/30	0.5205	0.3296	1.0	0.4688	0.5789	0.5
80/20	0.8372	0.3296	1.0	0.3810	0.5971	0.4286
90/10	0.4873	0.3296	0	0.5455	0.7101	0.5

Lastly, from the naive Bayes algorithm, the worst naive Bayes model performs on Wisconsin Breast Cancer with an accuracy of 0.5673 at a 50/50 train/test ratio, whereas the best naive Bayes model performs with 0.7101 accuracy at 90/10 (a 14.28% difference). On Somerville Happiness, the worst naive Bayes model performs with 0.5 accuracy at 70/30 and 90/10 train/test ratios, whereas the best naive Bayes model performs with 0.5703 accuracy at 20/80 (a 7.0312% difference).

4 DISCUSSION

Our remaining six experiments lead to the same conclusions: depending on the algorithm type and other factors such as train/test split and data sets, randomness has the potential to cause inconsistent and non-reproducible results. We also observe that in experiments where data is randomly assigned to training and testing sets, algorithms produce overall better performing (i.e., higher accuracy) models, suggesting a trade-off between model quality and reproducibility. The outcomes of ML models are clearly dependant upon the random numbers generated during training. Our findings show that the performance of a given ML model will vary significantly due to changes in the random number seed. Controlling the random numbers generated during training will regulate this variance in outcomes and ensure the reproducibility of ML models.

5 CONCLUSIONS AND FUTURE WORKS

The outcomes of a machine learning model are clearly dependant upon the random numbers generated during training. We have shown that the performance of a given machine learning model will vary significantly due to changes in the random number seed.

In this study, we examined the relationship between random number seed and model accuracy using common machine learning algorithms: neural networks, k-means clustering, and naive Bayes classification. Our findings show that, depending on the type of algorithm and other factors such as train/test split and data set, randomness has the potential to cause inconsistent and non-reproducible results. Controlling the random numbers generated during the training process will regulate this variance in outcomes and ensure the reproducibility of ML models.

We leave addressing other types of ML algorithms, parallel algorithms, and GPU support for future work as this paper is intended to offer a foundation to motivate extensive additional work.

6 ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

REFERENCES

[1] Susan Athey. 2019. 21. The Impact of Machine Learning on Economics. In *The economics of artificial intelligence*. University of Chicago Press, 507–552.

[2] Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature News* 533, 7604 (2016), 452.

[3] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. 2004. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD explorations newsletter* 6, 1 (2004), 20–29.

[4] David Ben-Israel, W Bradley Jacobs, Steve Casha, Stefan Lang, Won Hyung A Ryu, Madeleine de Lotbiniere-Bassett, and David W Cadotte. 2020. The impact of machine learning on patient care: a systematic review. *Artificial intelligence in medicine* 103 (2020), 101785.

[5] Stefano A Bini. 2018. Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care? *The Journal of arthroplasty* 33, 8 (2018), 2358–2361.

[6] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender systems survey. *Knowledge-based systems* 46 (2013), 109–132.

[7] Garry Choy, Omid Khalilzadeh, Mark Michalski, Synho Do, Anthony E Samir, Oleg S Pianykh, J Raymond Geis, Pari V Pandharipande, James A Brink, and Keith J Dreyer. 2018. Current applications and future impact of machine learning in radiology. *Radiology* 288, 2 (2018), 318–328.

[8] Dennis M Dimiduk, Elizabeth A Holm, and Stephen R Niezgoda. 2018. Perspectives on the impact of machine learning, deep learning, and artificial intelligence on materials, processes, and structures engineering. *Integrating Materials and Manufacturing Innovation* 7, 3 (2018), 157–172.

[9] Ronen Feldman. 2013. Techniques and applications for sentiment analysis. *Commun. ACM* 56, 4 (2013), 82–89.

[10] Paul Frederickson, Robert Hiromoto, and John Larson. 1987. A parallel Monte Carlo transport algorithm using a pseudo-random tree to guarantee reproducibility. *Parallel Comput.* 4, 3 (1987), 281–290.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[12] Neil P Chue Hong. 2021. Reproducibility Badging and Definitions: A Recommended Practice of the National Information Standards Organization. (2021).

[13] John PA Ioannidis. 2005. Why most published research findings are false. *PLoS medicine* 2, 8 (2005), e124.

[14] Philipp Koehn, Franz J Och, and Daniel Marcu. 2003. *Statistical phrase-based translation*. Technical Report. UNIVERSITY OF SOUTHERN CALIFORNIA MARINA DEL REY INFORMATION SCIENCES INST.

[15] Josua Krause, Adam Perer, and Kenney Ng. 2016. Interacting with predictions: Visual inspection of black-box machine learning models. In *Proceedings of the 2016 CHI conference on human factors in computing systems*. 5686–5697.

[16] Makoto Matsumoto and Takuji Nishimura. 1998. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. Model. Comput. Simul.* 8, 1 (Jan. 1998), 3–30. https://doi.org/10.1145/272991.272995

[17] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal* 5, 4 (2014), 1093–1113.

[18] Aditi Raghunathan, Sang Michael Xie, Fanny Yang, John C Duchi, and Percy Liang. 2019. Adversarial training can hurt generalization. *arXiv preprint arXiv:1906.06032* (2019).

[19] Yoram Reich. 1997. Machine learning techniques for civil engineering problems. *Computer-Aided Civil and Infrastructure Engineering* 12, 4 (1997), 295–310.

[20] Simone Scardapane and Dianhui Wang. 2017. Randomness in neural networks: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7, 2 (2017), e1200.

[21] J Ben Schafer, Joseph Konstan, and John Riedl. 1999. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*. 158–166.

[22] Koushik Sen. 2008. Race directed random testing of concurrent programs. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 11–21.

[23] Keng Siau and Yin Yang. 2017. Impact of artificial intelligence, robotics, and machine learning on sales and marketing. In *Twelve Annual Midwest Association for Information Systems Conference (MWAIS 2017)*. 18–19.

[24] Jonathan A Zdziarski. 2005. *Ending spam: Bayesian content filtering and the art of statistical language classification*. No starch press.

[25] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. Understanding deep learning (still) requires rethinking generalization. *Commun. ACM* 64, 3 (2021), 107–115.