Testing the Robustness of AutoML Systems

Tuomas Halvari Jukka K. Nurminen
Tommi Mikkonen
Department of Computer Science, University of Helsinki, Finland
first.[initial.]last@helsinki.fi

Automated machine learning (AutoML) systems aim at finding the best machine learning (ML) pipeline that automatically matches the task and data at hand. We investigate the robustness of machine learning pipelines generated with three AutoML systems, TPOT, H2O, and AutoKeras. In particular, we study the influence of dirty data on accuracy, and consider how using dirty training data may help create more robust solutions. Furthermore, we also analyze how the structure of the generated pipelines differs in different cases.

1 Introduction

Automated machine learning (AutoML) systems are used to find the best machine learning (ML) pipeline matching the task and data at hand, typically classification or regression. This includes model selection and hyperparameter optimization. Finding good models and hyperparameters are hard and time-consuming tasks for human experts, and they frequently involve a lot of trial-and-error experimentation. The promise of AutoML is that computers can automate these repetitive tasks and come up with good pipelines with little human effort. The drawback is that AutoML systems require a lot of computing power and the quality of the results varies. A recent overview of different AutoML systems [15] echoes these issues.

In this paper, we investigate the robustness of ML pipelines produced by AutoML mechanisms. Our focus is on user-friendly AutoML systems, which do not require prior knowledge about the data, algorithm choices, or hyperparameter spaces. For critical use cases, it is not enough that AutoML produces pipelines with accurate inference results. It is also important that the resulting pipelines tolerate faults, e.g. Gaussian noise, in data. At the moment AutoML is in an early phase and there seem to be no prior studies focusing on the robustness of their results. As AutoML gains maturity and ML systems are applied in safety-critical tasks deeper understanding of the robustness of the resulting systems is important. This paper is an early step towards that direction.

When building AI systems for robots and other autonomous devices, one consideration is their robustness against unexpected inputs, which commonly occur as a result of hardware or other problems in sensing and communication. Another class of unexpected inputs, which is outside of the scope of the present paper, is adversarial attacks, which aim for minimum input changes able to confuse the ML algorithms.

Recently there have been many papers comparing the performance of different AutoML systems [1, 15, 4]. Likewise, several papers discuss robust training of neural networks and vulnerability to adversarial inputs [9, 14, 10, 16]. Our study combines these two perspectives. More precisely, we measure the robustness of three different AutoML systems (TPOT [12], H2O AutoML [5], and AutoKeras [7]) with artificial inputs where we can control the type and amount of faults in the training and testing data. Our focus is on how dirty data, which arise if e.g. the camera of a robot is tilted or the lens is covered with dust, affects the performance. In particular, we focus on the following questions:

R.C. Cardoso, A. Ferrando, D. Briola, C. Menghi,
T. Ahlbrecht (Eds.): First Workshop on Agents and
Robots for reliable Engineered Autonomy 2020 (AREA'20).
EPTCS 319, 2020, pp. 103–116, doi:10.4204/EPTCS.319.8

- How accurate are the AutoML generated models when testing with dirty data (Section 4.1.1).
- What effect would training with on-purpose dirty data have on model accuracy? (Section 4.2.1).
- How similar/different pipelines do the different AutoML tools produce in the above cases? (Sections 4.1.2 and 4.2.2)
- How do the results vary as a function of the amount of faults in the testing data? (Sections 4.1 and 4.2)
- What AutoML systems to recommended for different use cases? (Section 4.3)

In the scope of this paper, we study the three AutoML systems (TPOT, H2O, AutoKeras) in the presence of data faults in training and/or testing data. Our experiments are built using dpEmu fault injector framework [11], which makes running such experiments easy. We control the amount of faults in the data and use two of the fault sources provided by dpEmu, namely Gaussian noise and rotation. Our testing focuses on image classification tasks. We use six different data fault levels with each data fault source and two different image datasets, namely Digits [13] and Fashion [17].

The structure of the paper is as follows. Section 2 gives background of AutoML systems. It also introduces the AutoML systems we study in this paper and the criteria for their selection. In Section 3, we discuss how the measurements were conducted and describe the used datasets and the faults generated to them. The results are presented in Section 4 and their meanings discussed in Section 5. Finally, we present our conclusions in Section 6.

2 AutoML systems

AutoML systems are meta-level machine learning algorithms, which use other ML solutions as building blocks for finding the optimal ML pipeline. In this context, an ML pipeline means the set of algorithms and their hyperparameters that the ML system uses to infer results from data. An AutoML system has to consider multiple ML pipelines and search values for their parameters. It needs to optimize each candidate pipeline to an adequate level but also ensure that enough time and resources are used to experiment with alternative pipelines. As a result, using AutoML systems can consume a lot of computing resources.

Typical tasks that many AutoML systems support are classification and regression. In various examples and benchmarks, typically image or text data are used. Some AutoML system like AutoKeras [7] even offer specialized image and text classifiers. Image data is usually easy to handle as a pixel array, with an integer value for each pixel, is used to represent each image. Pretty much all classification systems support this kind of input out of the box, and not much preprocessing is required. Unfortunately, this is not the case with text data, as it comes in many shapes. One dataset might be a list of strings and another a preprocessed dataset, where each string is represented as a sequence of integers representing the overall frequency in the data. While some AutoML systems like TPOT [12] and H2O AutoML [5] accept numerical arrays as inputs and do not care what the numbers represent, for example, AutoKeras has only specialized classifiers for image and text data. Because AutoKeras's text classifier only accepts text data as a list of strings and uses a built-in preprocessor, fair comparison to other more general AutoML systems may prove difficult. Therefore, we have left out the text data and only focus on image recognition in our study.

In this paper we study three different AutoML systems: TPOT, H2O, and AutoKeras. These were chosen because:

- All three provide a simple Python API and basic use requires only a few lines of code, which makes them easy to include in our benchmarks.
- These three are different enough in their approach to constructing the optimal ML pipeline. They also use different ML library backends.
- Unlike some of the available AutoML systems, with these three no previous knowledge of the data is required and no search space for the models and hyperparameters needs to be specified making them easy tools also for casual users.

Other free to use and open-source AutoML systems include MLBox [1] with required user-defined search spaces, auto-sklearn [3], which is similar to TPOT, and TransmogrifAI [2], which uses Apache Spark. Widely used cloud providers, such as Google Cloud, support AutoML [3], but, as part of the cloud business model, they are usually closed source and not free to use.

2.1 TPOT

TPOT [12] is a tree-based pipeline optimization tool. It uses the scikit-learn library [13] as the ML backend, and the classification models used include several models (Naive Bayes, Random Forest, Gradient Boosting, Linear SVC, Logistic Regression, etc.) from scikit-learn and the XGBoost classifier [2]. Aside from the actual ML models, the pipelines that TPOT creates can contain for example scalers, feature selection techniques, dimensionality reduction techniques, and other preprocessors [12]. TPOT uses genetic programming to evolve the pipeline sequence and hyperparameters to optimize certain criteria, like classification accuracy [12]. Inspecting the code reveals that TPOT uses predefined hyperparameter spaces for each model it considers. [4] TPOT offers no GPU support.

2.2 H2O

H2O AutoML [5] is a small and new part of the H2O.ai ML platform [5]. H2O's core code is written in Java, but a Python API is also provided. H2O AutoML supports the training of Stacked Ensemble models, which are collections of individual models. These Stacked Ensemble models are constructed by a meta learner called Super Learner [8] with a goal of combining a diverse set of different, base or optimized, models together.

The base models that H2O supports are Generalized Linear Models (GLM), Distributed Random Forests (DRF), XGBoost, Gradient Boosting Machines (GBM), and Deep Learning (NN). The hyperparameters used are chosen from a predefined search space using grid search. It seems that H2O chooses from 3 different options. It may use just one of the base models or their hyperparameter-optimized versions. It can also choose a Best Of Family Stacked Ensemble model, which includes one model from each category. The last option available is the All Models Stacked Ensemble pipeline, which can be very long. These three make up quite different choices for the best pipeline and in case of an easy dataset, with high accuracy.

Unlike the other two AutoML systems, H2O uses its own backend, which runs as a Java process. H2O offers a very limited GPU support: only XGBoost models can be trained with GPU, others are limited to CPU.

[1] https://github.com/AxeldeRomblay/MLBox
[2] https://transmogrif.ai/
[3] https://cloud.google.com/automl/
[4] https://github.com/EpistasisLab/tpot/blob/master/tpot/config/classifier.py
[5] https://www.h2o.ai/

2.3 AutoKeras

AutoKeras builds a deep learning neural model for your task and data. It optimizes both architecture and hyperparameters using neural network morphism guided by Bayesian optimization to select the most promising operations at each stage [7]. First, in each stage, the underlying model is trained with the proposed architecture and its performance is measured. Then, a new architecture is generated by optimizing an acquisition function. Finally, the performance of the new architecture is evaluated by training and testing the actual neural network. It uses Tensorflow, Keras and Torch backends. While TPOT and H2O do not support GPUs, AutoKeras offers full GPU support.

One of the interesting features of AutoKeras image classifier is the option to augment the train data to prevent overfitting and possibly increase robustness. [7] It uses random crops, random horizontal flips, and cutouts for data augmentation.

3 Research Approach

For performing the measurements, we have used parts of the dpEmu framework [11], a software framework for emulating common problems in data, testing the robustness of ML systems, and visualizing the results. The essential idea is that the system generates artificial faults to datasets according to predefined or user-defined fault models. The script run.py, that is used to run our benchmarks can be found in our repository [6]. It first creates different versions of the dataset at different data fault levels, given the data fault source. Then the benchmarked model is trained with different versions of the training data in a loop and after each step, the model is tested with all versions of the test data.

A total of twelve benchmarks were run for each model ranging from 15 mins to 6 hours. For the Digits dataset, the six benchmarks were 15 min, 30 min and 1 hour for both data fault sources. For the Fashion dataset, the six benchmarks were 1 hour, 3 hours and 6 hours for both data fault sources. This is the time available for each AutoML system to find and train the optimal classification pipeline. Notice that the images in the Digits dataset are too small in resolution for AutoKeras, so these results are unavailable.

3.1 Test setup

All the CPU-only benchmarks used in our testing were run on the University of Helsinki's Kale cluster using Intel Xeon E5-2680 v4 CPU's and a total of 40 cores with more than enough RAM. The benchmarks utilizing GPUs were run with a single Nvidia Tesla v100 GPU.

For TPOT and H2O we used the latest version available at the time of writing. For AutoKeras we used a slightly older but stable version 0.4.0, which may not have all the features of the newer versions, but enables us to set a time limit to benchmark the system properly with the others. The particular versions for the key components of our test system were: Python 3.7.0, Java 11.0.2, AutoKeras 0.4.0, H2O 3.28.0.3, TPOT 0.11.1, XGBoost 1.0.1, CUDA 10.0.130 and cuDNN 7.5.0.56.

3.2 Datasets

Two datasets, Digits and Fashion, of different sizes were used. The smaller dataset is the Digits dataset [13]. It consists of 1797 8x8 grayscale images of handwritten digits. The pixel values fall in range $0, \ldots, 16$. It was chosen because it is lightweight enough even for the heavier AutoML systems enabling them to optimize the pipelines more instead of just struggling to find a decent solution. The larger dataset

is the Fashion-MNIST [17] dataset, consisting of 70000 28x28 grayscale images of Zalando's articles. The pixel values are in the range $0, \ldots, 255$. It has been created as a more difficult replacement for the famous MNIST dataset, mainly because MNIST classification is too easy for modern ML algorithms. Like the Digits dataset, it contains images of articles from 10 different classes, 7000 each. With Digits and Fashion, $1/4$ and $1/7$ of the dataset were reserved for testing and the rest for training, respectively.

The key idea is to compare the large Fashion dataset with its big number of good training images with the small set of Digit training data. We especially want to see how fast the smaller training data becomes useless because of the lack of good training images when the amount of faults in the data increases.

3.3 Data fault sources

We used both Gaussian noise and image rotation as data fault types. Figure 1 shows examples of both fault types for both datasets. Both sources generate random faults. This means that even at high error levels it is possible but highly unlikely to get near original images. Notice also that the ranges of pixel values are different in the two datasets as described in Section 3.2. Six predefined data fault levels are used for both noise and rotation, including the clean level 0, with standard deviation and maximum angle as the data fault parameters respectively.

The reason for choosing these two was that while Gaussian noise effectively destroys parts of the information about the true label from the image, whereas, at least for the human eye, rotating the image makes little difference to the shape of its object.

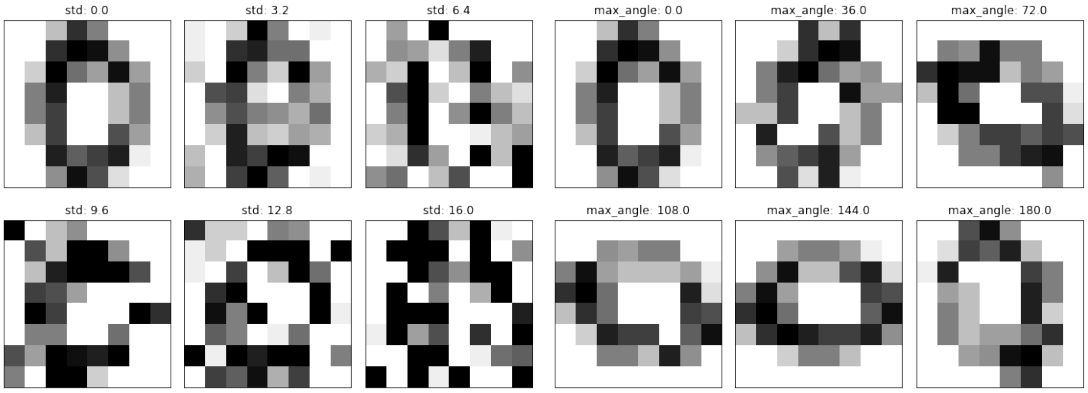
3.4 Metrics

Our primary metric for image classification is the accuracy score when comparing predicted and true labels. The accuracy score was chosen over the F1-score because there are no imbalanced classes in either of the datasets.

3.5 Validity and limitations

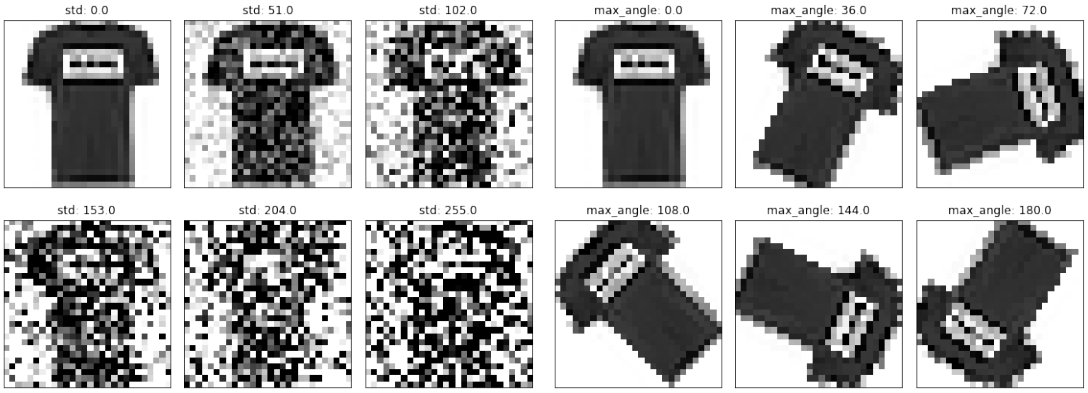
To make the comparison fair, the following points have been considered:

- We focus on image classification because, unlike text processing, it is done in rather similar ways in all three systems. Text data would have required preprocessing for TPOT and H2O while AutoKeras would automate it.
- We conduct the testing using time limit based categories in a way similar to other studies [4]. Otherwise, the runtimes would vary greatly and the results would be more dependent on the default parameters.
- We allocated all computing resources to work on one model with one fault type at a time.
- We allowed only some of the initialization parameters to be fixed. Such parameters include time limits, random seeds, the parameters enabling the model to use more CPUs or RAM, and the parameters used to modify logging output.
- AutoKeras benefits significantly from the GPU use and has a optional feature for image augmentation. Thus we used three different versions for AutoKeras, namely CPU, GPU and GPU with image augmentation.



(a) Digits image with different levels of Gaussian noise

(b) Digits image with different levels of random rotation



(c) Fashion image with different levels of Gaussian noise

(d) Fashion image with different levels of random rotation

Figure 1: Example images from Digit and Fashion datasets using different data fault sources at different data fault levels

The accuracy of AutoKeras GPU version results varied a lot form one run to the other in comparison to the other tested systems. Therefore, we report the accuracy results for AutoKeras as average over two runs for each benchmark.

Though the two image dataset are quite different regarding to both image and dataset sizes, they are still quite similar with single item in each image. In a future work, a more realistic dataset with colors could also be included.

4 Results

In this Section, unless otherwise mentioned, for noise source std is fixed at the second level, meaning 6.4 for the Digits dataset and 102 for the Fashion dataset. For rotation source maximum angle is fixed at the last data fault level corresponding to 180 degrees for both datasets, meaning all possible rotations are equally likely. Complete results are available at our GitHub repository [6].

Table 1: Summary of best accuracy results per model for both datasets, given the training and testing data fault sources at a fixed data fault level.

Dataset	Digits							Fashion						
Training data	Clean			Noise		Rotation		Clean			Noise		Rotation	
Testing data	Clean	Noise	Rotation	Clean	Noise	Clean	Rotation	Clean	Noise	Rotation	Clean	Noise	Clean	Rotation
AutoKeras CPU	-	-	-	-	-	-	-	0.914	0.205	0.250	0.836	0.819	0.820	0.807
AutoKeras GPU	-	-	-	-	-	-	-	0.925	0.283	0.244	0.833	0.812	0.839	0.829
AutoKeras GPU with Aug.	-	-	-	-	-	-	-	0.945	0.159	0.278	0.744	0.851	0.881	0.877
H2O	0.987	0.676	0.289	0.973	0.842	0.887	0.838	0.905	0.449	0.233	0.838	0.821	0.805	0.792
TPOT	0.987	0.887	0.373	0.951	0.838	0.891	0.853	0.882	0.492	0.236	0.801	0.782	0.760	0.760

Table 2: Summary of the effect of time to accuracy when both training and testing with clean data.

	Digits			Fashion		
Benchmark	15 min	30 min	1 h	1.5 h	3 h	6 h
AutoKeras CPU	-	-	-	0.887	0.912	0.912
AutoKeras GPU	-	-	-	0.908	0.921	0.916
AutoKeras GPU with Aug.	-	-	-	0.928	0.933	0.930
H2O	0.984	0.986	0.982	0.902	0.902	0.905
TPOT	0.985	0.985	0.987	0.876	0.879	0.882

4.1 How good are AutoML generated models with clean training data?

4.1.1 Accuracies

Let us start by comparing the five AutoML systems when both training and testing is done with clean data. The accuracy results for both datasets can be found in Table 1. They report the maximum accuracy of the six runs with different maximum execution times (typically longer execution times improved the results but not always, see Tables 2 and 4).

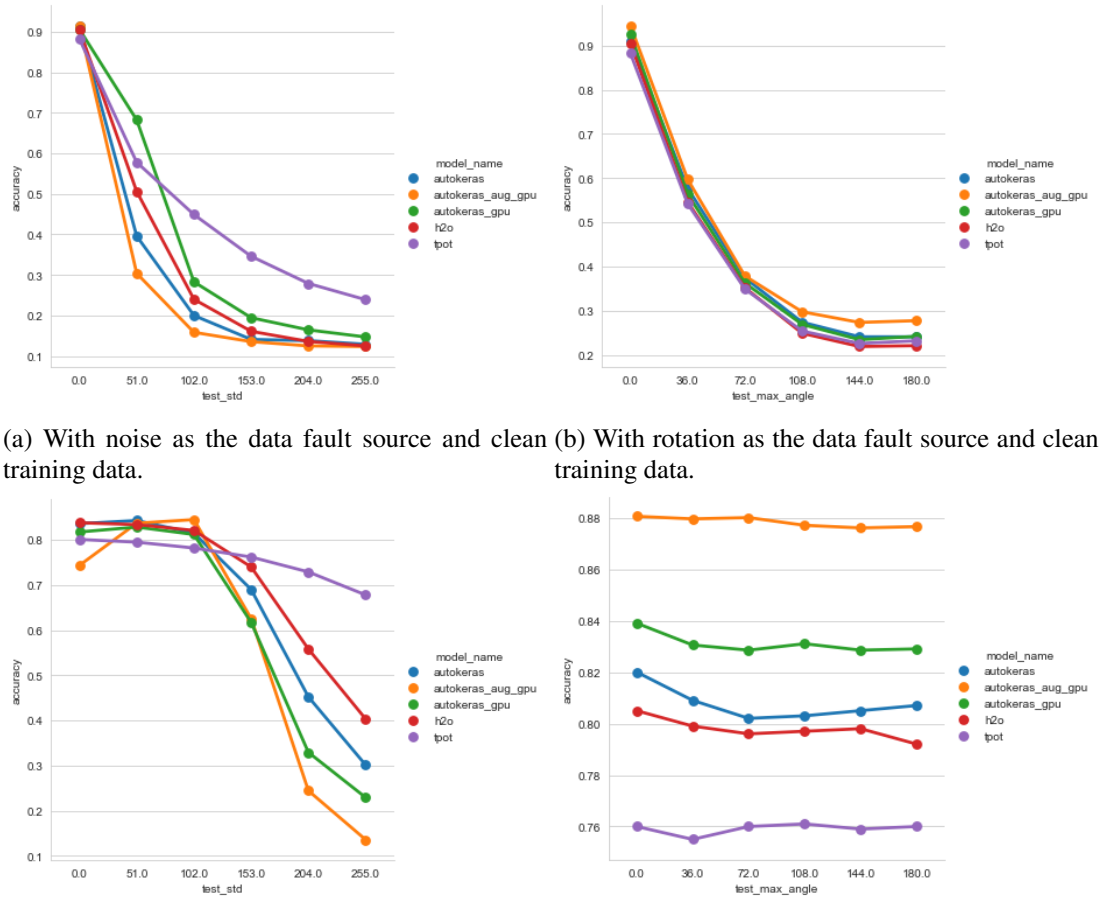
Let's for now focus just on the columns with clean training and testing data. For the Digits dataset, it seems that both H2O and TPOT are equally good. On the other hand, with the Fashion dataset, all AutoKeras versions seem strong and data augmentation seems to help with accuracy. H2O seems to beat TPOT slightly.

The results for the effect of benchmarking time to the accuracy when both training and testing with clean data can be seen in Table 2. When looking at the accuracy transitions from 1.5 h to 3 h with the larger Fashion dataset, AutoKeras CPU and AutoKeras GPU with image augmentation seem to require more time to reach the optimal performance, when compared to the other three test cases. Especially the CPU version of AutoKeras seems to struggle in creating a neural network with CPU resources only. When testing, AutoKeras CPU with image augmentation enabled did not seem viable at all so GPU training seemed to be the only option.

Let's then move our focus to the columns with clean training data and dirty testing data in Table 1. With the Digits dataset, TPOT seems to beat H2O when the test data fault source is noise and also when it's rotation. Thus it is interesting to see that the accuracies are pretty even with the Fashion dataset although in Figure 2a there seems to be a wide difference. We have to remember that the values in Table 1 are the best values among the three benchmarks with noise as the data fault source for each model. The difference can be explained by looking at Table 3. As we can see, in shorter Fashion benchmarks with noise as the data fault source, H2O sometimes has only time to run the XGBoost models, which seems to give better accuracies at higher test data fault levels with noise as the data fault source [6], thus being more robust than the longer Stacked Ensemble pipelines. With the small Digits dataset, H2O seemed to

Table 3: Summary of the effect of time to the best pipelines per model given the training data. The results in parentheses are from alternative runs.

	Training	Digits			Fashion		
	data	15 min	30 min	1 h	1.5 h	3 h	6 h
AutoKeras	Clean	-	-	-	CNN 16 (16) layers	CNN 66 (66) layers	CNN 67 (67) layers
	Noise	-	-	-	CNN 16 layers	CNN 66 layers	CNN 67 layers
	Rotation	-	-	-	CNN 16 layers	CNN 66 layers	CNN 483 layers
AutoKeras GPU	Clean	-	-	-	CNN 69 (483,483,483) layers	CNN 68 (69,70,77) layers	CNN 68 (70,483,483) layers
	Noise	-	-	-	CNN 68 (69) layers	CNN 483 (483) layers	CNN 80 (483) layers
	Rotation	-	-	-	CNN 483 (483) layers	CNN 71 (483) layers	CNN 77 (79) layers
AutoKeras GPU with Aug.	Clean	-	-	-	CNN 66 (66,66,66) layers	CNN 69 (69,70,483) layers	CNN 72 (79,483,483) layers
	Noise	-	-	-	CNN 66 (66) layers	CNN 66 (67) layers	CNN 483 (483) layers
	Rotation	-	-	-	CNN 66 (66) layers	CNN 483 (483) layers	CNN 71 (483) layers
H2O	Clean	StackedEnsemble BestOfFamily 6 models (AllModels 75 models)	StackedEnsemble BestOfFamily 6 models (AllModels 192 models)	StackedEnsemble AllModels 296 (341) models	XGBoost (StackedEnsemble AllModels 6 models)	XGBoost (StackedEnsemble AllModels 15 models)	StackedEnsemble AllModels 25 (30) models
	Noise	StackedEnsemble AllModels 72 models	StackedEnsemble AllModels 91 models	StackedEnsemble AllModels 135 models	StackedEnsemble SE AllModels 3 models	StackedEnsemble BestOfFamily 4 models	StackedEnsemble AllModels 28 models
	Rotation	StackedEnsemble BestOfFamily 6 models			XGBoost	StackedEnsemble AllModels 4 models	StackedEnsemble AllModels 24 models
TPOT	Clean	LogisticReg. +DT clf +KNN clf (same)	RF clf+2 models +KNN clf (GB clf+2 models +KNN clf)	GB clf +KNN clf (same)	RF clf (same)		
	Noise	KNN clf		MultinomialNB+ KNN clf	LinearSVC		OneHotEncoder +KNN clf
	Rotation	GB clf +RF clf +KNN clf	ET clf +KNN clf	GB clf +5 models +KNN clf	KNN clf		RF clf



(a) With noise as the data fault source and clean training data.

(b) With rotation as the data fault source and clean training data.

(c) With noise as the data fault source and dirty training data.

(d) With rotation as the data fault source and dirty training data.

Figure 2: Accuracy plots for the 6 h benchmark with the Fashion dataset when testing at different data fault levels given the data fault source and training data.

have ample time.

Moving again to the Fashion dataset, in Table 1 AutoKeras' different versions seem to perform worse when compared to H2O and TPOT when the data fault source is noise. Especially AutoKeras GPU with image augmentation enabled shows poor performance. We can see in Figure 2a that this is true even at the higher test data fault levels. In the same Table, all of the benchmarked systems seem to perform equally with rotation as the data fault source. Though in Figure 2b we can see that AutoKeras GPU with image augmentation seems to pull ahead of the competition at higher test data fault levels. This is probably due to that the image augmentation process includes some rotations, as mentioned in Section 2.3.

4.1.2 Pipelines

The results for the optimal pipelines can be found in Table 3. When inspecting only the rows with clean training data we can see a few things. Looking at the pipelines for AutoKeras' different versions

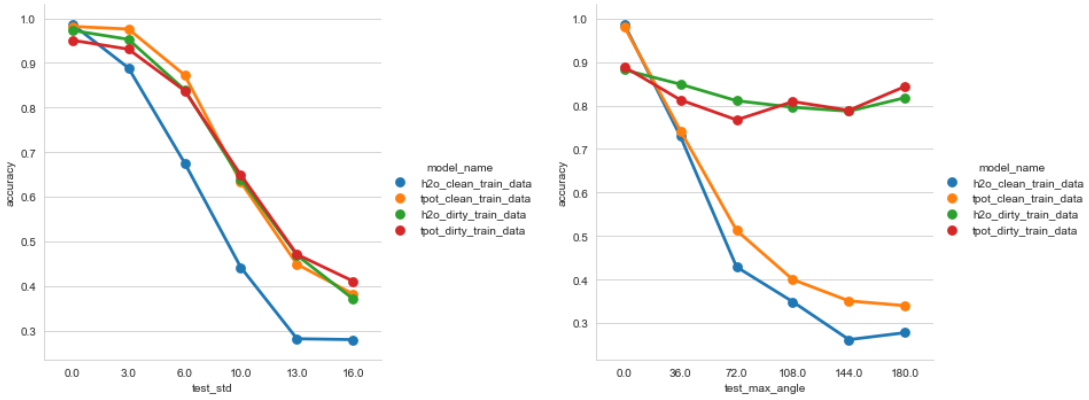
and the model summaries in the repo [6], we can see that it prefers very similar pipelines in different benchmarks, which are shared even between different versions of AutoKeras used in our tests. It also seems to mainly use one of three base pipelines of different lengths. As the training time increases, the possible modifications to these pipelines and the hyperparameters seem to appear at the end, and with the longer benchmarks, we can see a few layers being added to the end of the base pipelines. Looking at H2O's pipelines we can see that the length of the pipeline varies even more than with AutoKeras, ranging from 6 to 341 models for the Digits dataset and from 1 to 30 models for the Fashion dataset.

For H2O, we can see some of the Stacked Ensemble models explained in Section 2.2 and some pipelines based on a single model. The presence of a single XGBoost classifier among the StackedEnsemble models can be explained by looking at the H2O log files in our repo [6], which show that H2O moves to the other base models discussed in Section 2.2 only after all base XGBoost models have been tested. Also, the training of all the XGBoost models takes most of the runtime. So if the benchmark time is limited, the XGBoost classifier could be the only option. For TPOT the choice of the dataset seems to affect the chosen pipeline. With the Digits dataset, TPOT seems to like the K-Neighbors classifier. With the Fashion dataset, Random Forest classifier seems to be the only choice.

4.2 How good are AutoML generated models with dirty training data?

4.2.1 Accuracies

Let's first consider the columns with dirty training data and clean testing data in Table 1. With the Digits dataset, H2O and TPOT seem to perform quite similarly with both data fault sources when training with dirty data, using parameters explained in the beginning of Section 4. With the Fashion dataset and rotation as the data fault source, AutoKeras GPU with image augmentation is the clear winner as can be seen in Table 1, but unfortunately seems to be the worst with noise.



(a) With noise as the data fault source.

(b) With rotation as the data fault source.

Figure 3: Accuracy plots for the 1 h benchmark with the Digits dataset when testing at different data fault levels given the data fault source.

Let's then compare the five AutoML systems when both training and testing is done with dirty data. With the Digits dataset, H2O and TPOT seem to perform quite similarly with both data fault sources when testing with dirty data at different data fault levels, as can be seen in Figures 3a and 3b with maybe H2O having a slight edge at mid-levels when rotation is the source.

Table 4: Summary of the effect of time to accuracy when both training and testing with dirty data.

	Training data fault source	Digits			Fashion		
		15 min	30 min	1 h	1.5 h	3 h	6 h
AutoKeras CPU	Noise	-	-	-	0.782	0.819	0.815
	Rotation	-	-	-	0.725	0.795	0.807
AutoKeras GPU	Noise	-	-	-	0.807	0.812	0.812
	Rotation	-	-	-	0.810	0.821	0.829
AutoKeras GPU with Aug.	Noise	-	-	-	0.837	0.851	0.845
	Rotation	-	-	-	0.860	0.848	0.877
H2O	Noise	0.842	0.840	0.840	0.796	0.799	0.821
	Rotation	0.820	0.838	0.818	0.781	0.788	0.792
TPOT	Noise	0.827	0.827	0.838	0.776	0.776	0.782
	Rotation	0.818	0.853	0.844	0.737	0.737	0.760

Table 5: Summary of the optimal pipelines for TPOT given the dataset and training data fault source and level.

Dataset	Training data fault source	Data fault level					
		0 (clean)	1	2	3	4	5
Digits	Noise	GB clf +KNN clf	KNN clf	MultinomialNB +KNN clf	MultinomialNB	MultinomialNB +KNN clf	ET clf +MultinomialNB
	Rotation	GB clf +KNN clf	GB clf +MultinomialNB +KNN clf	GB clf +KNN clf	GB clf +KNN clf	RF clf +ET clf +KNN clf	GB clf +5 models +KNN clf
Fashion	Noise	RF clf	XGB clf	OneHotEncoder +KNN clf	LinearSVC	LinearSVC + GB clf	LinearSVC
	Rotation	RF clf					

When looking at the same results for the Fashion dataset and rotation in Figure 2d, we can see that the accuracies do not really drop as the data fault level increases as all possible rotations are covered with the huge set of training data. AutoKeras GPU with image augmentation seems to be the clear winner here while TPOT clearly performs the worst. With noise, all AutoKeras versions seem to struggle at higher data fault levels, where TPOT seems to excel, as can be seen in Figure 2c. Furthermore, AutoKeras GPU with image augmentation has a peculiar performance. The accuracy on the test data seems to peak at the level that was used on the training data, clearly lacking robustness with bad scores at both ends.

The results for the effect of benchmarking time to the accuracy when both training and testing with dirty data can be seen in Table 4. With the Digits dataset, while H2O's scores seem to have stabilized after 15 min, TPOT might need more time to reach the optimal results. With the Fashion dataset, 1.5 h clearly is not enough for AutoKeras CPU. This can also be seen from Table 3, which shows that after 1.5 h, the CNN has only 16 layers. The rest of the tested systems show minor improvements with time.

4.2.2 Pipelines

When comparing the pipelines that the tested systems produce, we can see from Table 3, and from the model summaries in the repo [6], that the general pipelines for AutoKeras' versions and H2O do not change that much, even though H2O has its issues with the large dataset and short benchmark time. With TPOT, the preferred pipelines tend to change a lot more based on the dataset and the data fault source. With the smaller Digits dataset, if rotation is the source, TPOT seems to favor the K-Neighbors classifier as part of the pipeline as can be seen in Table 5. This is also true with noise as the source if the data fault level is low. With higher levels of noise in the training data, Multinomial Naive Bayes seems to be the preferred choice. Regarding the larger Fashion dataset, with noise as the data fault source, Logistic

Regression seems to be the model of choice at higher data fault levels. On the other hand, with rotation, TPOT seems to use a Random Forest classifier at all levels.

4.3 Recommendations and comparison

To begin with, the following recommendations for different use cases can be made, based on the results above:

- When both training and testing with clean data, AutoKeras GPU with image augmentation seems to be the clear winner but requires a lot of time and computing power.
- When training with clean data and testing with dirty data, due to the inconsistencies of H2O with shorter training times, TPOT would be the optimal choice when noise is the data fault source. With rotation, AutoKeras GPU with image augmentation would be the top choice because of good performance with both clean and very faulty test data.
- When training with dirty data and testing with clean data, while AutoKeras GPU with image augmentation is the clear winner with rotation, whereas with noise there is no clear winner. H2O seems to best TPOT with both datasets, but equal the performance of the two other AutoKeras versions with Fashion.
- When both training and testing with dirty data, TPOT seems to be the winner when noise is the data fault source because of its constantly good performance even at high data fault levels. AutoKeras GPU with image augmentation is once again the clear winner with rotation.

Given the test data fault source, with the larger datasets like Fashion, where good training images are plenty, training with dirty data is in most cases the better option with both data fault sources as can be seen when comparing the plots for each model in Figures 2a and 2c for noise, and Figures 2b and 2d for rotation. This is also the case with the much smaller Digits dataset when using rotation as the data fault source, as can be seen in Figure 3b. However, with noise, training with dirty data is not necessarily the best option as can be seen in Figure 3a. In fact, training with dirty data seems to be the clear winner only in H2O's case. With TPOT the models seem to perform quite similarly at the mid and higher data fault levels. The other exception to this rule, is obviously when we know that the test data is clean.

4.4 Resource usage

There were several differences in the resource usage between the three AutoML systems. With 40 cores AutoKeras used almost 100% of the CPU resources available and around 8 GB of RAM. The two GPU versions of AutoKeras used almost exclusively GPU and the same amount of memory.

H2O's CPU usage was around 80% It typically used around 150GB with the larger Fashion dataset when given 350GB of memory to use for the Java process. Some of the runs failed due to a segmentation fault and had to be rerun.

We noticed that TPOT had trouble parallelizing some of the models it uses, and CPU efficiency was around 30%. We observed times when only one core's usage was maxed out. Regarding the RAM usage, for TPOT around 30 GB was usually enough.

5 Discussion

The authors of Fashion-MNIST report accuracy scores, with a similar train-test split, for several classifiers and state-of-the-art Neural Networks (NNs). [6] The best reported score was 0.897 for basic classifiers (SVC) and 0.967 for state-of-the-art NNs (WRN40-4). Looking at our best results with clean test data in Table 1, even with dirty training data our best results compare quite well.

Regarding TPOT's memory usage mentioned in Section 4.4, the official documentation of TPOT includes a warning of possible memory issues when multiple cores are used [12]. We too noticed occasional peaks to around 200 GB when testing with 96 cores. Others [7] have also noticed issues with ML systems when using too many cores.

To explain the results from Section 4.3 related to the similar performance of models trained with clean or dirty data with the smaller Digits dataset and noise, we have to consider the nature of the data fault sources. When using data fault source like rotation, most information in the image is retained in the data even at higher data fault levels. However, a data fault source like Gaussian noise destroys parts of the information in the image, as can be seen in Figure 1a. Because the Digits dataset is small and the random nature of Gaussian noise, we are left with only a few good training images, so training with dirty data may not be the best choice after all. Also, the effects of Gaussian noise are particularly noticeable with the Digits dataset, because we are using very low-resolution images as training data. Regarding the larger Fashion dataset, there are still plenty good training images within the huge training dataset.

As for the results in 4.1 regarding AutoKeras' bad performance with clean training data and dirty test data, this is a known problem for neural networks as discussed in Section 1. In the image augmentation enabled version's case, it could be said that because of the random crops, horizontal flips, and cutouts discussed in Section 2.3, the neural network becomes even more sensitive to certain data fault sources destroying information from the data.

It is also known that AutoML systems can recommend alternative optimal solutions in different runs for the same problem [8]. We also observed this. The cause may be the tight time limits imposed on a system with stochastic elements or just that two pipelines offer almost equal performance.

6 Conclusions

Based on the results, using training data, which contains examples of faults the system will encounter is promising: accuracy with clean test data drops a bit but robustness increases a lot. We also noted that different AutoML systems produce very different ML pipelines. TPOT even generated rather different pipelines for clean, noisy, and rotated data.

Future work of exploring if our findings apply to not so similar datasets and different data fault sources is important. Future AutoML tools may want consider robustness as an explicit optimization goal. Perhaps the user could specify preferred trade-off between accuracy and robustness.

References

[1] Adithya Balaji & Alexander Allen (2018): *Benchmarking Automatic Machine Learning Frameworks*. *ArXiv* abs/1808.06492.

[6] `https://github.com/zalandoresearch/fashion-mnist`
[7] `https://github.com/szilard/GBM-multicore`
[8] `https://epistasislab.github.io/tpot/using/`

[2] Tianqi Chen & Carlos Guestrin (2016): *XGBoost*. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, doi:10.1145/2939672.2939785.

[3] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum & Frank Hutter (2019): *Auto-sklearn: Efficient and Robust Automated Machine Learning*, pp. 113–134. Springer International Publishing, Cham, doi:10.1007/978-3-030-05318-5_6.

[4] P. J. A. Gijsbers, Erin LeDell, Janek Thomas, S'ebastien Poirier, Bernd Bischl & Joaquin Vanschoren (2019): *An Open Source AutoML Benchmark*. *ArXiv* abs/1907.00909.

[5] H2O.ai (2017): *H2O AutoML*. Available at `http://docs.h2o.ai/h2o/latest-stable/h2o-docs/automl.html`. H2O version 3.30.0.1.

[6] Tuomas Halvari: *AutoML comparison*. Available at `https://github.com/thalvari/AutoML_comparison`.

[7] Haifeng Jin, Qingquan Song & Xia Hu (2019): *Auto-Keras: An Efficient Neural Architecture Search System*. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, Association for Computing Machinery, New York, NY, USA, p. 1946–1956, doi:10.1145/3292500.3330648.

[8] Mark J. van der Laan, Eric C Polley & Alan E. Hubbard (2007): *Super Learner*. *Statistical Applications in Genetics and Molecular Biology* 6(1), doi:10.2202/1544-6115.1309. Available at `https://www.degruyter.com/view/journals/sagmb/6/1/article-sagmb.2007.6.1.1309.xml.xml`.

[9] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras & Adrian Vladu (2018): *Towards Deep Learning Models Resistant to Adversarial Attacks*. *ArXiv* abs/1706.06083.

[10] S. Moosavi-Dezfooli, A. Fawzi & P. Frossard (2016): *DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks*. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2574–2582, doi:10.1109/CVPR.2016.282.

[11] Jukka K Nurminen, Tuomas Halvari, Juha Harviainen, Juha Mylläri, Antti Röyskö, Juuso Silvennoinen & Tommi Mikkonen (2019): *Software Framework for Data Fault Injection to Test Machine Learning Systems*. In: *Proceedings of 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE 2019) Workshops*, IEEE, pp. 294–299, doi:10.1109/ISSREW.2019.00087.

[12] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz & Jason H. Moore (2016): *Evaluation of a Tree-Based Pipeline Optimization Tool for Automating Data Science*. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, Association for Computing Machinery, New York, NY, USA, p. 485–492, doi:10.1145/2908812.2908918.

[13] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot & Édouard Duchesnay (2011): *Scikit-Learn: Machine Learning in Python*. *J. Mach. Learn. Res.* 12(null), p. 2825–2830, doi:10.5555/1953048.2078195.

[14] Uri Shaham, Yutaro Yamada & Sahand Negahban (2018): *Understanding adversarial training: Increasing local stability of supervised models through robust optimization*. *Neurocomputing* 307, pp. 195 – 204, doi:10.1016/j.neucom.2018.04.027. Available at `http://www.sciencedirect.com/science/article/pii/S0925231218304557`.

[15] A. Truong, A. Walters, J. Goodsitt, K. Hines, C. B. Bruss & R. Farivar (2019): *Towards Automated Machine Learning: Evaluation and Comparison of AutoML Approaches and Tools*. In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1471–1479, doi:10.1109/ICTAI.2019.00209.

[16] Tsui-Wei Weng, Pin-Yu Chen, Lam M. Nguyen, Mark S. Squillante, Akhilan Boopathy, Ivan Oseledets & Luca Daniel (2019): *PROVEN: Verifying Robustness of Neural Networks with a Probabilistic Approach*. In: *Proceedings of the 36th International Conference on Machine Learning*, *Proceedings of Machine Learning Research* 97, pp. 6727–6736.

[17] Han Xiao, Kashif Rasul & Roland Vollgraf (2017): *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. *ArXiv* abs/1708.07747.