

BRIAN MINGUS / MODEL EVALUATION

Benchmarking AutoML.

Hyper-MNIST

Synthetic data / June 2022



TECHNICAL REPORT | JUNE 2022

Evaluating Off-The-Shelf ML Systems with Hyper-MNIST

Brian Mingus

Cambium Assessment

A transformed MNIST benchmark explores memorization, generalization and extrapolation when evaluating off-the-shelf machine-learning systems.

Collection cover added September 2026. Original manuscript follows unchanged.
AI-generated conceptual artwork; separate from the scientific figures.

brianmingus.com

Evaluating Off-The-Shelf ML Systems with Hyper-MNIST

Brian Mingus
Machine Learning Team
Cambium Assessment

June 2022

1 Introduction

We wish to evaluate the machine learning systems we have access to, including proprietary / commercial systems such as DataRobot. Our target domain includes data covered under FERPA, and is explicitly disallowed by such commercial systems, yet such systems provide relatively comprehensive baselines against which we can evaluate the systems we can use in-house, such as Auto-Sklearn [link] and Auto-PyTorch [link], and allow us to take direction from our peers in industry. Since we can't use in-house data on third-party systems, we must use a proxy dataset. Working in automated scoring, a good choice of proxy dataset might be a linguistic dataset, such as the Automated Essay Scoring competition on Kaggle [link]. However, this dataset is rather large and will have a long runtime, incurring significant cost on DataRobot, which appears to be extremely expensive. Furthermore, DataRobot likely includes this data in its automated evaluation systems. A similar argument applies to MNIST. Although MNIST is not generally used as a NLP dataset, it is fairly high dimensional, with 10 classes, 784 features, 60,000 training instances and 10,000 testing instances, and it continues to drive new breakthroughs, recently in the area of convex hull extrapolation [4], and also in 2001 [1].

Here, we present a modified version of MNIST that these systems have never seen before, and evaluate their performance on it. Our MNIST derivation supports memorization, generalization and extrapolation, and an early evaluation with Auto-PyTorch achieved test set generalization of 92% accuracy and 1.15 RMSE, leaving plenty of room for improvement, assuming improvement is possible.

2 Hyper-MNIST

To construct a variant of MNIST, with MNIST denoted as X and the target vector denoted as y , we start with a data-driven prior, namely, the sum of the

vectors in each class, denoted W , with row-major dimensions of $(10, 784)$. We choose W as the variable for the sums to indicate that the sum of the vectors for each class will be used as weights in a linear regression $\hat{Y} = XW^T$. $\hat{Y}$ is capitalized to indicate that the output of XW^T results in a matrix of decisions, currently with dimensions of $(60000, 10)$. Next, we compute the classifier decision for each of 60,000 instances using $\text{argmax}(\hat{Y})$, and create a new dataset derived from X consisting of the residuals (errors) from the first pass, which we denote R . R is the subset of X that W classified incorrectly.

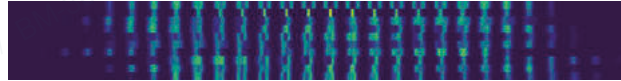
We then repeat this procedure using the residuals, R , as our dataset, resulting in the equation $\hat{Y}_R = RW_R^T$. Indeed, we recursively repeat the procedure a total of seven more times, as such: $\hat{Y}_{R_{1..8}} = R_{1..8}W_{R_{1..8}}^T$, and we stack the outputs, resulting in a matrix of dimension $(60000, 80)$ which we denote H , which is Hyper-MNIST.

A visual explanation is in order.

We first take the sum of each class, and stack them on top of each other. Here, you can see the sum of each class reshaped back to images and concatenated left-to-right.



When stacked on top of each other we obtain W^T , our weights, seen below. These weights are a sensible starting point for a linear model such as a linear regression or a Support Vector Machine (SVM). Technically, these models include an intercept or bias term, resulting in the interpretation that the weights constitute hyperplanes of a dimension one larger than the dataset, which allows one to separate the data by drawing lines (hyperplanes) between the classes. To our knowledge, including a bias term has not been proven to help performance, and it may be omitted. Since we omit the bias, these weights may be better interpreted as Lagrange Multipliers [2], the same multipliers as above are seen here without reshaping, with the sum of zeros on the first row, and the sum of 9s on the bottom row.

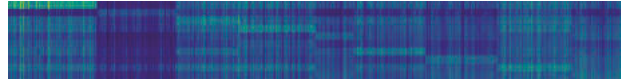


We then multiply these weights times X , which is MNIST, as $\hat{Y} = XW^T$, or more simply, $\hat{Y} = XW$, with X having dimension $(60000, 784)$ and W having dimension $(784, 10)$, resulting in a matrix $\hat{Y}$ with dimension $(60000, 10)$, seen here:

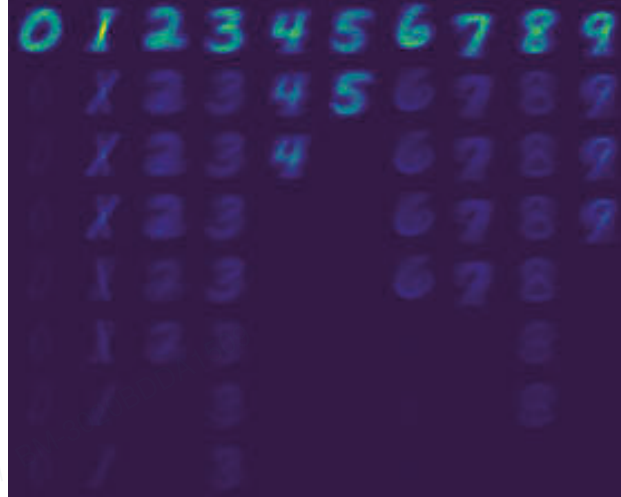


To create the above image we first sorted the images in MNIST (X) by class. The weight vectors are also sorted by class. Thus, there is substantial vertical and horizontal banding, as the dataset $\hat{Y}$ represents raw model decisions, and the model is doing a decent job.

We next determine whether argmax over the decisions of the rows of $\hat{Y}$ actually works with the code $\text{sum}(eq(\text{argmax}(X@W, 1), y))$ which tells us that this simple model gets 40310/60000 correct, for an accuracy of 67.18% and RMSE of 2.695. From here we can generate a new version of $\hat{Y}$ containing only rows where argmax returns the correct response:

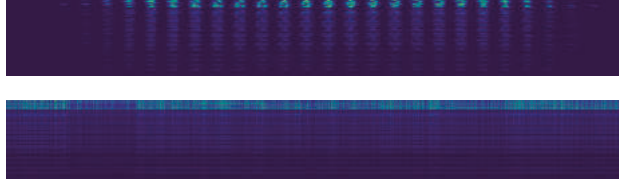


We can do the same for the incorrect responses, and from which we can derive our first set of weights based on the sum of instances in each class in the residuals, R , which is the second row below, W_2 :



Thus, in the above, the first row is the sum of each class in MNIST, W_1 , the second row is the sum of each class in the residuals, W_2 , the third row is the sum of each class in the residuals of the residuals, W_3 and so on. The number of instances remaining correctly classified by each set of weights in turn is 40310, 5433, 3416, 3893, 3820, 1394, 617, 584 for a total of 59467 / 60000 correct on the train set. If we continue recursing, we can obtain 100% on the train set [3].

Rendered as $W_{R_{1..8}}$ with dimension $(784, 80)$ we have the first image below, and if we multiply $XW_{R_{1..8}}$ we obtain H or Hyper-MNIST, the second image:



Combined with the class labels, an outstanding empirical question is whether Hyper-MNIST maintains mappable variance that corresponds to the variance in MNIST, to what extent, and whether that mappable variance is essentially isomorphic to MNIST itself. In the future, we can test whether many randomly sampled Lagrange Multipliers, W_L result in a more evaluative dataset, H_L , than H .

3 Results

In the following results we compare the performance of AutoPyTorch trained on all 60,000 instances in H and tested on the 10,000 instance MNIST holdout set ($X_{Holdout}W_R$) with a variety of models from DataRobot, each trained on 38,400 instances in the training set, and each generalizing to the remaining 21,600 instances in the training set.

Model	Holdout RMSE
Keras Residual Cross Network Regressor Using Training Schedule (3 Cross Layers, 4 Layers: 100, 100, 100, 100 Units)	.8387
Advanced GLM Blender	.8359
ENET Blender	.8471
Advanced AVG Blender	.8559
eXtreme Gradient Boosted Trees	.9870
Keras Residual AutoInt Regressor	1.0166
Gradient Boosted Trees Regressor	1.1197
Light Gradient Boosted Trees Regressor	1.1270
Keras Slim Residual Neural Network Regressor	1.1483
AutoPyTorch(H)	1.15
ExtraTrees Regressor	1.1708
Nystroem Kernel SVM Regressor	1.252
Random Forest Regressor	1.2551
LightGBM Random Forest Regressor	1.3672
Keras Wide Residual Neural Network Regressor	1.3995
Keras Deep Self-Normalizing Residual Neural Network Regressor	1.5201
Decision Tree Regressor	1.6022
Ridge Regressor with Binned Numeric Features	1.7183
Generalized Additive Model	1.7394
RuleFit Regressor	1.7070
Eureqa Regressor	1.8911
Dropout Additive Regression Trees Regressor	1.9074
Ridge Regression	1.9416
Elastic-Net Regressor	1.9914
Lasso Regressor	1.9914
Linear Regression	1.9945
Auto-tuned Stochastic Gradient Descent Regression	2.2144
$\text{argmax}(\hat{Y})$ (MNIST train set)	2.695
Mean Response Regressor	2.8738

Preliminary results suggest that our data-driven prior W_R maps much of the variance in MNIST, and yet still presents a formidable challenge, with .8387 RMSE still remaining. It seems likely that the RMSE would go down substantially by allowing DataRobot to train on the entire train set, using samples

from Infinite MNIST to boost its size somewhat. Additionally, it seems likely that some of the information may have been washed out by the choice of prior, which would turn out to not be representative of all the variance in the dataset. On the other hand, it seems, perhaps even more likely, that this problem can be completely solved, even with the restricted training set. Indeed, this problem is most likely not challenging enough. It seems worthwhile to replace W with a single random vector, v , resulting in a vector of decisions that must be disentangled.

References

- [1] P. Haffner. “Escaping the Convex Hull with Extrapolated Vector Machines”. In: *Neural Information Processing Systems* (2001).
- [2] R. Mamen. “Constrained Optimization Using a Conjugate Gradient Technique, with Control Applications”. In: *Journal of Optimization Theory and Applications* (1972).
- [3] D Wolpert. “Stacked Generalization”. In: *Neural Networks* (1992).
- [4] R. Yousefzadeh and F. Huang. “Using Wavelets and Spectral Methods to Study Patterns in Image-Classification Datasets”. In: *arXiv* (June 2020).