Comparison of Analytical Techniques for the MNIST handwritten digits

George Fisher *

July 21, 2015

Abstract

The US Post Office's desire to automate the routing of mail by handwritten zipcode motivated the creation of the MNIST database of handwritten digits [9], [14]. The database contains 60,000 training images and 10,000 testing images, each 28×28 grayscale images of a single digit, and is widely used for benchmarking machine learning algorithms, the best of which is reported to have achieved a 0.23% misclassification error rate using convolutional neural networks [8], [2]. Kaggle [7] has a training contest using a variation of the MNIST database. This paper describes the results of my attempts to beat the benchmarks set for the various algorithms and then my results on Kaggle.

Contents

*George escaped from a 30-year international life of crime on Wall Street, got a Masters degree in quantitative finance from MIT, climbed Kilimanjaro and (some of) Everest and as of 2015 is pursuing an interest in machine learning. george at georgefisher dot com

1 Introduction

The purpose of this exercise was to take a real dataset that had been thoroughly analyzed and benchmarked by serious people and attempt to do at least as well as the published benchmarks with the same algorithms plus a few others that have recently become popular. In the process I hoped to learn the algorithms very well, as well as the models implementing the algorithms with their various parameters, and to get experience using them in a pseudo-production environment. The final benchmark I set for myself was to use what I learned to perform well on the Kaggle Digit Recognizer contest leaderboard.

MNIST Database The files are kept in a zipped, binary format. Procedures to read the unzipped binary files in Python and R can be found in Appendix **??** and **??**.

Hardware I have an 8-core Intel i7 32GB Zareason Linux laptop running Ubuntu 15.04. I also used AWS EC2 servers extensively: see this excellent YouTube video: `https://youtu.be/NQu3ugUkYTk` to learn how to configure an RStudio server; see [1] and [13] for instructions on setting up a GPU server for deep learning; see [15] for instructions for setting up an iPython server.

Programming Environment R version 3.2.0 [12], Python 2.7.9, iPython 3.1.0 [11], H2O and OpenCV 3.0.0 were the primary programming environments used, using RStudio Version 0.99.441 and jupyter 3.1 (iPython notebooks). In R I used `data.table`s because of their greater efficiency. This document uses LaTeX contained in an Sweave/knitr document on RStudio.

Process For each algorithm I ran 5-fold cross-validation on the training set to determine the best model parameters and then I created a model using the parameters that produced the lowest misclassification rate on the full training set and predicted the full test set, which had not been used in any way prior to this fitting. This was done for for the original and the deskewed data.

Source code The code for this project can be found on GitHub at `https://github.com/grfiv/MNIST` [4].

Acknowledgements Dan Weiner of Boston University taught me Mathematical Statistics and Mathematical Analysis; these plus Linear Algebra are at the heart of modern data analysis. Jerome Friedman of Stanford University taught me to look at the algorithms in a rigorous mathematical way rather than viewing them as mysterious black boxes. Bill Howe of the University of Washington introduced me to this whole field with his Coursera course `Introduction to Data Science`; at the end of his course he said that those who graduated with distinction could consider themselves *advanced beginners*, which both insulted and motivated me ... I hope to have come a little way further than that by now.

I stand on the shoulders of many giants. The benchmarks were set in 1998 and much has changed since then:

- the **theory** has improved: Friedman, Breiman et al published their seminal works around 2000; ESL [5] was first published in 2001.
- the **hardware** has improved: Moore's Law is still going strong and the un-named law for storage is stronger, still.
- the **processes** have improved: parallel processing has made great strides and Google's pre-Hadoop breakthrough paper [3] was published in 2004; R and Python have become tremendously effective; Amazon Web Service has re-introduced the old-fashioned idea of computer timesharing on an enormous scale and at a cost that anyone can afford.
- the **popularity** of the field has soared ('Big Data', 'Data Science') bringing the power of crowd sourcing with it through forums like Kaggle.

All of these advantages gave me an edge in beating the benchmarks: the tools I used and the algorithms I employed simply did not exist then.

2 Deskewing & csv files

The benchmark results were almost uniformly better when reading deskewed digits. OpenCV has a deskewing algorithm [10] which I implemented using an iPython (jupyter) notebook to deskew the entire database.

3 Single-hidden-layer neural networks

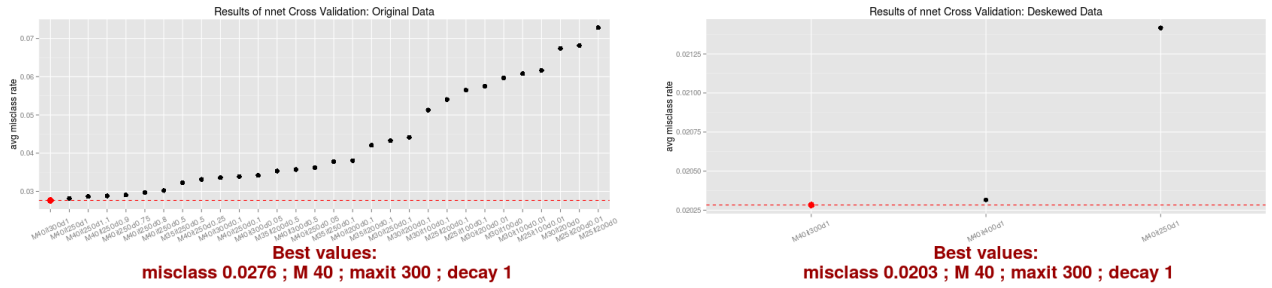


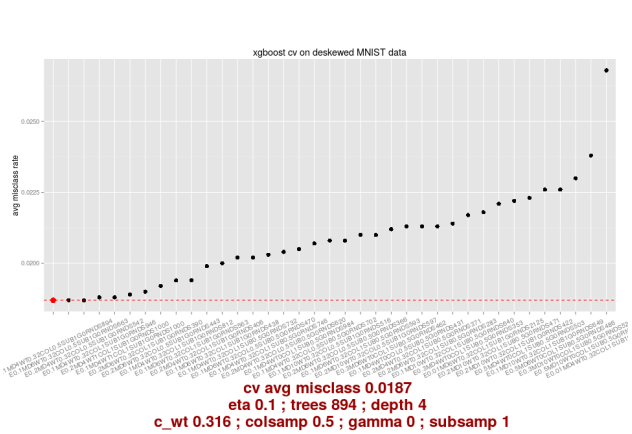
Figure 1: `nnet` 5-fold Cross Validation

	package	language	preprocessing	misclass	benchmark	parameters
1	avNNet	R	deskew	0.0126	0.0840	M=100,maxit=250,decay=1.0,repeats=3
2	nnet	R	none	0.0167	0.1200	M=100,maxit=250,decay=0.5

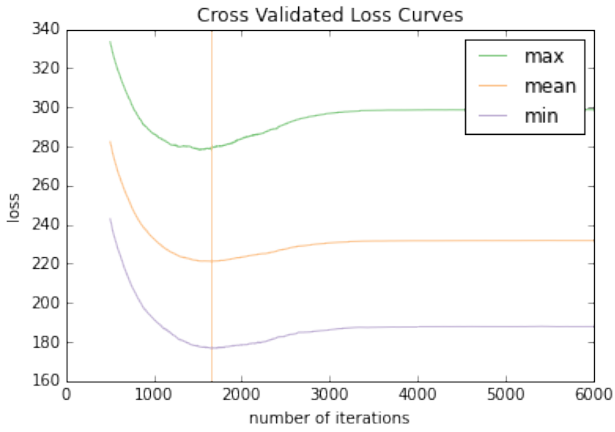
Table 1: Model full training set, predict test set

The limiting constraint for parameterizing `nnet` was RAM: 40 hidden nodes using 5-fold CV with each fold running on its own core took 29Gb, which was pretty-much the upper limit of my laptop; increasing `maxit` added time but the process could just be set to run in the background. Once I saw how the parameterization was going and realized that 40 hidden nodes outperformed any fewer for any given combination of `maxit` and `decay`, I optimized `decay` using the original training data and then optimized `maxit` using the deskewed training data. For the purpose of the benchmark, running a single model once, a `M` of 100 took 27Gb. I could have dispensed with running the CV in parallel but it would have taken 5 times as long and it ran plenty long as it was; I did make use of AWS to get more RAM and as a place to send long-running tasks.

4 Boosted trees



xgboost



scikit-learn gbm

Figure 2: 5-fold Cross Validation

	package	language	preprocessing	misclass	benchmark	parameters
1	gbm	H2O	deskewed	0.0159	0.0126	eta=0.1;trees=894,depth=4
2	xgboost	R	deskewed	0.0171	0.0126	eta=0.1;trees=894,depth=4
3	gbm	scikit-learn	deskewed	0.0260	0.0126	eta=0.01;trees=1650,depth=4

Table 2: Model full training set, predict test set

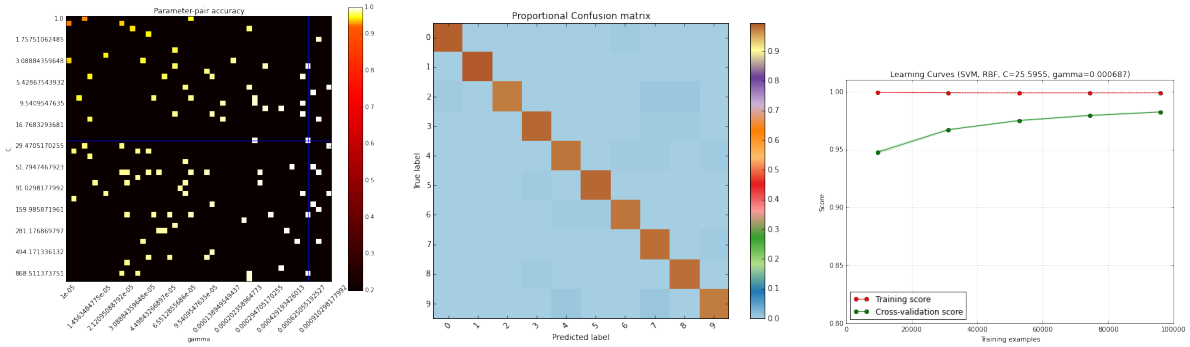
Boosted trees are widely recognized as being excellent classifiers, possibly the best 'out of the box' ensemble methods: generally better than either bagged trees or random forests, and `xgboost` is touted as being far superior to the old standby of `gbm`.

It is true that `xgboost` runs quickly and its memory use is quite conservative: barely more than 6Gb while running cross validation on the entire MNIST training dataset. There are a lot of parameters to tune and that takes a lot of time although the early stopping feature makes it possible to leave the specification of the number of trees out of the grid.

H2O is currently getting a lot of press but as of June 2015 it had neither cross validation nor grid search capabilities for its `h2o.gbm` model so I used parameters found with the other models. H2O was very good with storage: less than 4Gb.

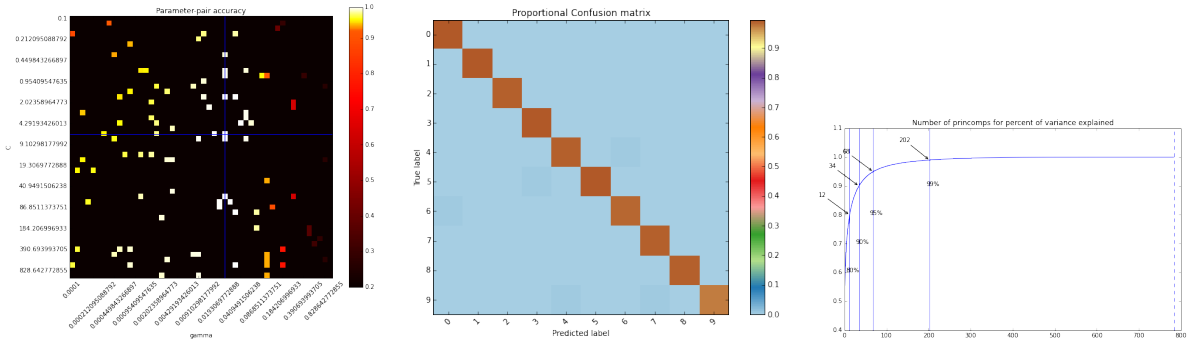
R's `gbm` is a memory pig: out-of-memory fitting 2,000 trees in 122Gb. Python's scikit-learn `GradientBoostingClassifier` was much more frugal.

5 Support Vector Machines



random parameter fit | confusion matrix | learning curves

Figure 3: Support Vector Machine: RBF Kernel



random parameter fit | confusion matrix | variance explained

Figure 4: Support Vector Machine: RBF Kernel & PCA

The RBF learning curves show a continuous, if decreasing improvement in prediction from adding more data. These curves result from training on both the original and the deskewed training data and predicting the deskewed test data, 120,000 observations in total; the shapes of the (green) curve seems to indicate that the prediction would improve from training on even more obsevations.

	package	language	preprocessing	misclass	benchmark	parameters
1	SVC	scikit-learn	PCA 85%	0.0103	0.0140	rbf;C=2.947;gamma=0.015999
2	SVC	scikit-learn	PCA 85%	0.0123	0.0140	poly;C=1.0;gamma=0.1112;degree=3;coef0=1
3	SVC	scikit-learn	both	0.0182	0.0140	rbf;C=25.596;gamma=0.00069
4	SVC	scikit-learn	deskewed	0.0221	0.0140	rbf;C=25.596;gamma=0.00069
5	OpenCV SVM	Python	deskew, HOG	0.0560	0.0140	

Table 3: Model full training set, predict test set

6 Partial Least Squares

For multi-class classification the accepted approach for PLS seems to be to apply LDA to the best number of principal components. I'd like to extract the components themselves as predictor variables but I haven't found a way to do it; `sklearn.cross_decomposition.PLSSVD` seems to offer a way but I got terrible results.

	package	language	preprocessing	misclass	benchmark	parameters
1	pls.lda	R	deskew	0.0875		
2	pls caret	R	deskew	0.0985		ncomp=29

Table 4: Model full training set, predict test set

7 Vowpal Wabbit

8 K-Nearest Neighbors

9 Multi-Layer Neural Networks

10 Ensembles

11 Summary

	package	language	preprocessing	misclass	benchmark	parameters
1	SVC	scikit-learn	PCA 85%	0.0103	0.014	rbf;C=2.947;gamma=0.015999
2	SVC	scikit-learn	PCA 85%	0.0123	0.014	poly;C=1.0;gamma=0.1112;degree=3;coef0=1
3	avNNet	R	deskew	0.0126	0.084	M=100,maxit=250,decay=1.0,repeats=3
4	gbm	H2O	deskewed	0.0159	0.0126	eta=0.1;trees=894,depth=4
5	nnet	R	none	0.0167	0.12	M=100,maxit=250,decay=0.5
6	xgboost	R	deskewed	0.0171	0.0126	eta=0.1;trees=894,depth=4
7	SVC	scikit-learn	both	0.0182	0.014	rbf;C=25.596;gamma=0.00069
8	SVC	scikit-learn	deskewed	0.0221	0.014	rbf;C=25.596;gamma=0.00069
9	gbm	scikit-learn	deskewed	0.0260	0.0126	eta=0.01;trees=1650,depth=4
10	OpenCV SVM	Python	deskew, HOG	0.0560	0.014	
11	pls.lda	R	deskew	0.0875		
12	pls caret	R	deskew	0.0985		ncomp=29

Table 5: Model full training set, predict test set

12 Kaggle Contest

The Digit Recognizer 'Getting Started' contest [6] provides a 42001 x 785 csv training dataset with a header row, the first column, called 'label', is the digit that was drawn by the user; and a 28001 x 784 csv test dataset with a header row but no label column. 42000+28000 == 60000+10000, so this looks a lot like the MNIST dataset rearranged. There are many ways to game the contest since the entire MNIST dataset is labeled, but I played it straight: I created the model with the training data we were given and predicted the test set without peeking.

	algorithm	preprocessing	parameters	result
1	scikit.svc	PCA 85%	rbf;C=2.947;gamma=0.015999	0.989
2	scikit.svc	PCA 85%	poly;C=1.0;gamma=0.1112;degree=3;coef0=1	0.988
3	xgboost	both	eta=0.1;trees=894,depth=4	0.98129
4	xgboost	deskewed	eta=0.1;trees=894,depth=4	0.98014
5	scikit.svc	both	rbf;C=25.596;gamma=0.00069	0.97786
6	avNNet	deskew	M=100,maxit=250,decay=1,repeats=3	0.97129
7	scikit.gbm	deskewed	eta=0.01;trees=1650,depth=4	0.96957
8	h2o.gbm	deskewed	eta=0.1;trees=894,depth=4	0.969
9	nnet	deskew	M=100,maxit=250,decay=0.5	0.96029
10	nnet	none	M=100,maxit=250,decay=0.5	0.94386

Table 6: Kaggle Results

References

[1] Markus Beissinger. How to install Theano on Amazon EC2 GPU instances for deep learning. http://markus.com/install-theano-on-aws/, April 2015.

[2] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification. Computer Vision and Pattern Recognition, February 2012. CVPR 2012, p. 3642-3649.

[3] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters . http://research.google.com/archive/mapreduce.html, 2004.

[4] George Fisher. MNIST: Initial setup plus nnet R algorithm, June 2015.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2011.

[6] Kaggle. Digit Recognizer. https://www.kaggle.com/c/digit-recognizer, 2015. accessed June 2015.

[7] Kaggle. Kaggle: The home of data science. https://www.kaggle.com/, 2015.

[8] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings - IEEE*, November 1998.

[9] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/, 1998. accessed June 2015.

[10] OpenCV. OCR of Hand-written Data using SVM. https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_ml/py_svm/py_svm_opencv/py_svm_opencv.html. OpenCV 3.0.0-dev.

[11] Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, May 2007.

[12] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[13] Adrian Rosebrock. Deep Learning on Amazon EC2 GPU with Python and nolearn. http://www.pyimagesearch.com/2014/10/13/deep-learning-amazon-ec2-gpu-python-nolearn/, October 2014.

[14] Wikipedia. MNIST database — Wikipedia, The Free Encyclopedia, 2015. [Online; accessed 10-June-2015].

[15] Randy Zwitch. Cluster Computing for $0.27/hr using Amazon EC2 and IPython Notebook. http://badhessian.org/2013/11/cluster-computing-for-027hr-using-amazon-ec2-and-ipython-notebook/, November 2013.