# Comparison of Analytical Techniques for the MNIST handwritten digits

George Fisher *

August 4, 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 [5], [8]. The database contains 60,000 training images and 10,000 testing images, each $28 \times 28$ 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 [4], [1]. Kaggle 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 on GitHub.

**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 [10] for instructions for setting up an iPython server.

**Programming Environment** R 3.2.1, RStudio 0.99.467, Python 2.7.9, iPython 3.1.0 were the primary programming environments. This document uses LaTeX contained in an Sweave/knitr document on RStudio.

**Process** Where possible I ran 5-fold cross-validation on the training set, employing a random grid search to determine the model parameters that produced the lowest average misclass rate on the held-out folds. Then I predicted the test set with the best model.

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

**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 [3] 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 [2] 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.

# 2 Expanding the training dataset

The benchmark results were almost uniformly better when reading deskewed digits. OpenCV has a deskewing algorithm [7] which I implemented using an iPython (jupyter) notebook to deskew the entire database. There's a project called The infinite MNIST dataset [6] which provides software (which I did not use) to deform the images to create a larger training set.

# 3 Single-hidden-layer neural networks

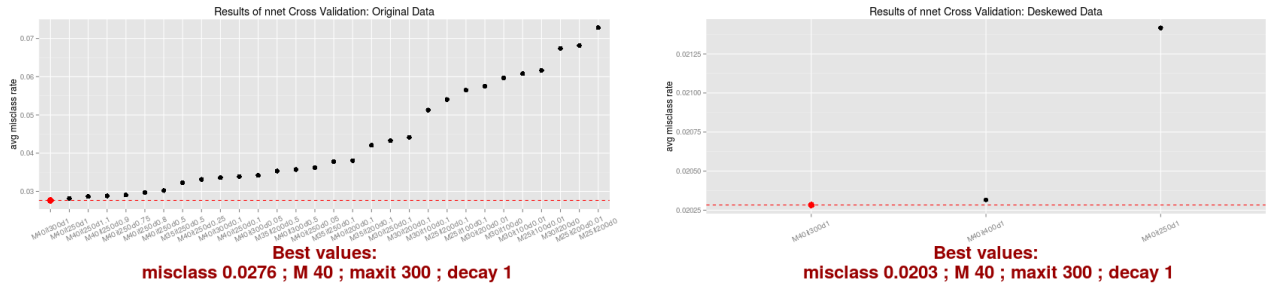


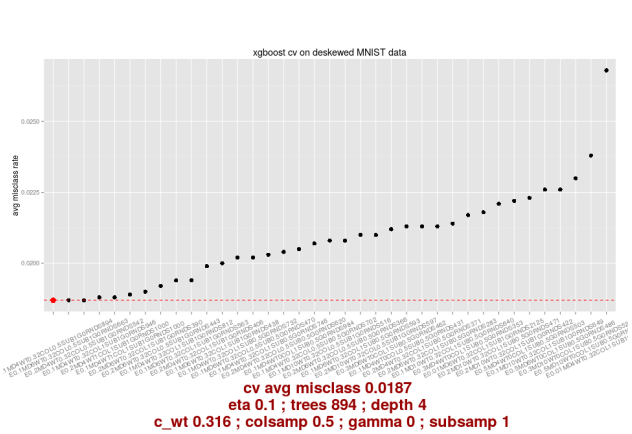

Figure 1: `nnet` 5-fold Cross Validation

| | 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 |

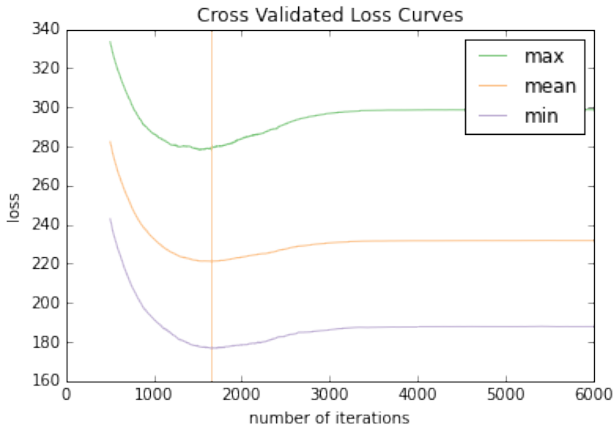
Table 1: Model full training set, predict test set

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

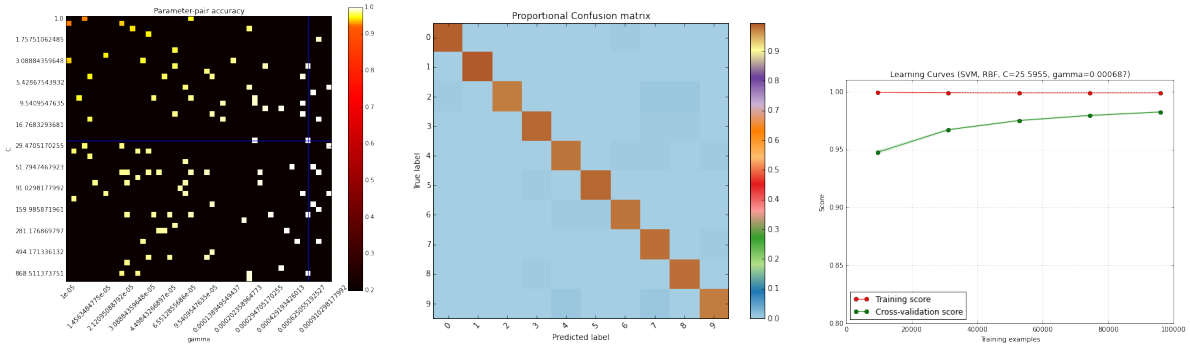
Boosted trees are widely recognized as being excellent classifiers, possibly the best 'out of the box' ensemble methods: generally better than either bagged trees or random forests, and `xgboost` is touted as being far superior to the old standby of `gbm`.

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.

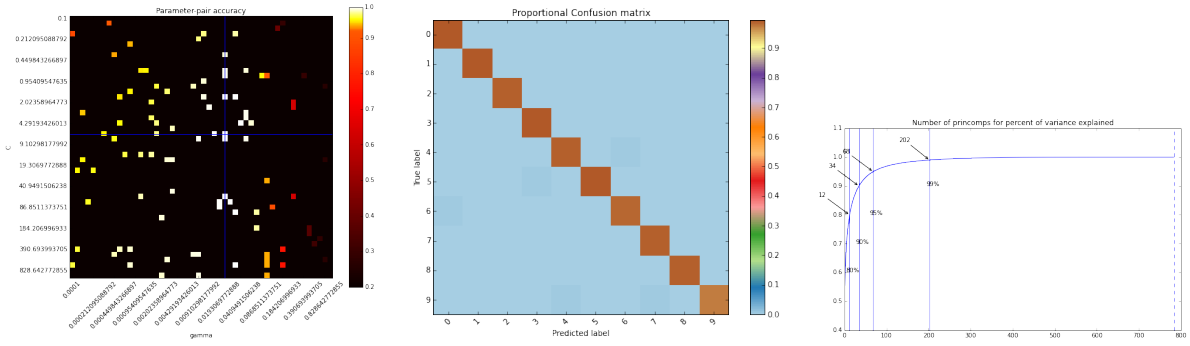
R's `gbm` is a memory pig: out-of-memory fitting 2,000 trees in 122Gb. Python's scikit-learn `GradientBoostingClassifier` was much more frugal.

# 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

This was my first look at VW. Everybody is cheering about how good it is, 'the Swiss Army knife of ML', etc. It may indeed be good but overly-documented it ain't; a useful tutorial is [9]. VW is fast and doesn't use very much memory at all, both of which are very much in its favor and it does have early stopping so you can specify a large number of passes over the data without worrying about what number to give. A lot of Googling got me snippets of useful code like parameter estimation (crude, no CV that I can see, keywords with 'holdout' in their name but no explanation; no random grid search except what you can hand craft yourself), plus neural nets, SVM and boosted trees but it was a bit of a struggle to find even that much. Owen Zhang says 'Enables fast iterative development of features and model structure', but I do not know how to do this.

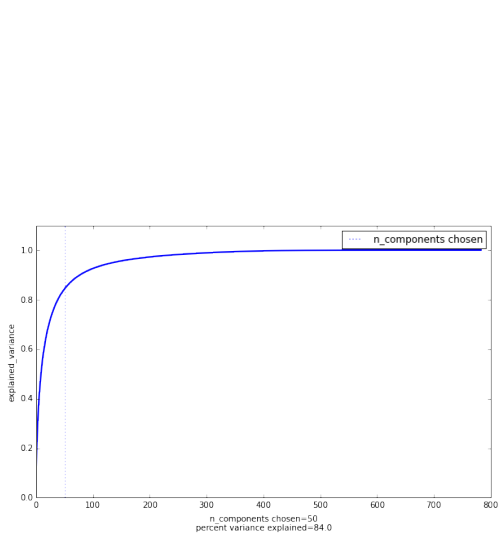
| | package | language | preprocessing | misclass | benchmark | parameters |
|---|---|---|---|---|---|---|
| 1 | vw | vw | original | 0.0135 | | –oaa 10 -q ii |
| 2 | vw | vw | pca | 0.0170 | | –oaa 10 -q ii |
| 3 | vw | vw | original | 0.0176 | 0.084 | –nn 100 |

Table 5: Model full training set, predict test set

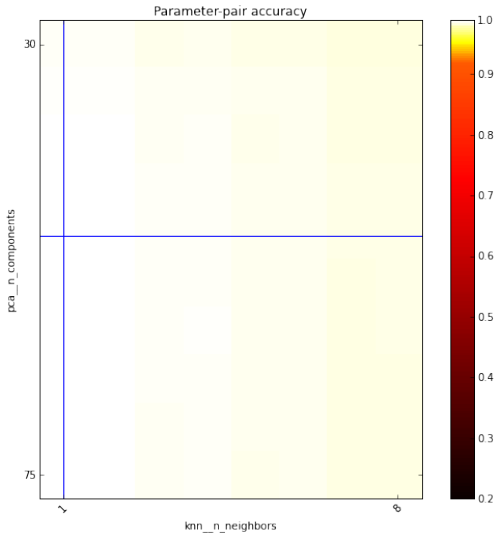
# 8 K-Means

I couldn't get anything useful out of 10-means.

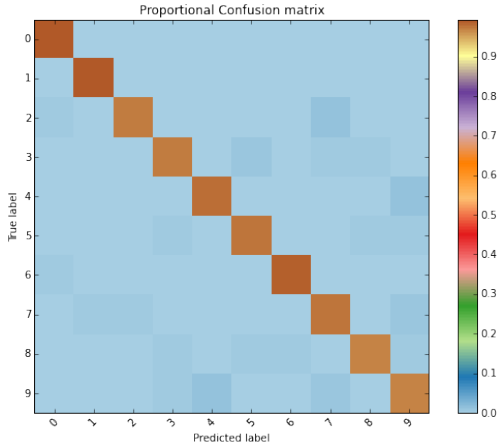
# 9 K-Nearest Neighbors



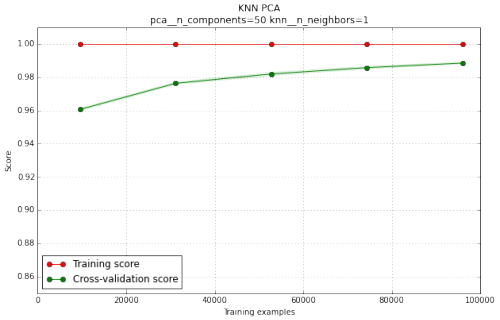

PCA spectrum




grid search results




test results




learning curves

Figure 5: K Nearest Neighbors & PCA

I don't know why my learning curves achieve a better result than the confusion matrix produced from the predictions.

| | package | language | preprocessing | misclass | benchmark | parameters |
|---|---|---|---|---|---|---|
| 1 | knn | scikit | PCA | 0.0207 | | weights='distance',n_neighbors=1,n_components=50 |
| 2 | knn | scikit | both | 0.0226 | | weights='distance',n_neighbors=1 |

Table 6: Model full training set, predict test set

# 10 hyperopt

| | package | language | preprocessing | misclass | benchmark |
|---|---|---|---|---|---|
| 1 | hyperopt svc | hyperopt-sklearn | PCA | 0.0153 | n_components=96;poly;C=0.092;coef0=0.534;degree=3;g |
| 2 | hyperopt svc | hyperopt-sklearn | PCA | 0.0173 | rbf |

Table 7: Model full training set, predict test set

# 11 Fourier Analysis

I have had a nagging feeling that Fourier analysis could be useful but the following image from Explicit feature map approximation for RBF kernels sort of took the wind out of my sails:

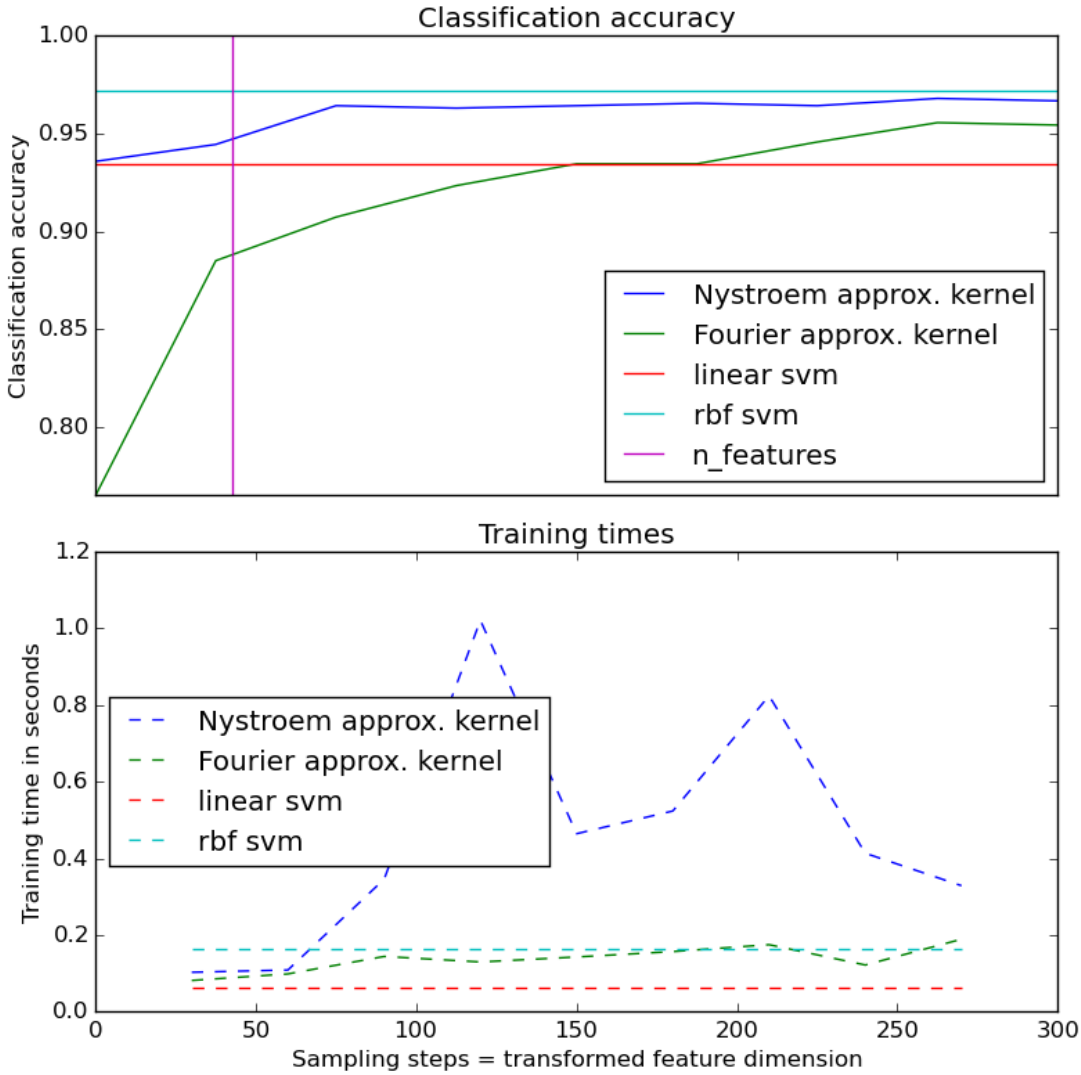



Figure 6: Comparison of RBF and Fourier kernels with SVM

# 12 Multi-Layer Neural Networks

# 13 Ensembles

# 14 Summary

| | package | language | preprocessing | misclass | benchmark | parameters |
|---|---|---|---|---|---|---|
| 1 | theano MLP | python | PCA | 0.0092 | | |

Table 8: Model full training set, predict test set

| | package | language | preprocessing | misclass | benchmark |
|---|---|---|---|---|---|
| 1 | theano MLP | python | PCA | 0.0092 | |
| 2 | SVC | scikit-learn | PCA 85% | 0.0103 | 0.014 |
| 3 | SVC | scikit-learn | PCA 85% | 0.0123 | 0.014 |
| 4 | avNNet | R | deskew | 0.0126 | 0.084 |
| 5 | vw | vw | original | 0.0135 | |
| 6 | hyperopt svc | hyperopt-sklearn | PCA | 0.0153 | n_components=96;poly;C=0.092;coef0=0.534;degree= |
| 7 | gbm | H2O | deskewed | 0.0159 | 0.0126 |
| 8 | nnet | R | none | 0.0167 | 0.12 |
| 9 | vw | vw | pca | 0.0170 | |
| 10 | xgboost | R | deskewed | 0.0171 | 0.0126 |
| 11 | hyperopt svc | hyperopt-sklearn | PCA | 0.0173 | rbf |
| 12 | vw | vw | original | 0.0176 | 0.084 |
| 13 | SVC | scikit-learn | both | 0.0182 | 0.014 |
| 14 | knn | scikit | PCA | 0.0207 | |
| 15 | SVC | scikit-learn | deskewed | 0.0221 | 0.014 |
| 16 | knn | scikit | both | 0.0226 | |
| 17 | gbm | scikit-learn | deskewed | 0.0260 | 0.0126 |
| 18 | OpenCV SVM | Python | deskew, HOG | 0.0560 | 0.014 |
| 19 | pls.lda | R | deskew | 0.0875 | |
| 20 | pls caret | R | deskew | 0.0985 | |

Table 9: Model full training set, predict test set

# 15 Kaggle Contest

The Digit Recognizer 'Getting Started' contest 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 | theano MLP | deskewed | | 0.99257 |
| 2 | scikit.svc | PCA 85% | rbf;C=2.947;gamma=0.015999 | 0.989 |
| 3 | scikit.svc | PCA 85% | poly;C=1.0;gamma=0.1112;degree=3;coef0=1 | 0.988 |
| 4 | scikit.knn | pca | n_components=49,n_neighbors=1 | 0.98229 |
| 5 | xgboost | both | eta=0.1;trees=894,depth=4 | 0.98129 |
| 6 | xgboost | deskewed | eta=0.1;trees=894,depth=4 | 0.98014 |
| 7 | scikit.svc | both | rbf;C=25.596;gamma=0.00069 | 0.97786 |
| 8 | avNNet | deskew | M=100,maxit=250,decay=1,repeats=3 | 0.97129 |
| 9 | scikit.gbm | deskewed | eta=0.01;trees=1650,depth=4 | 0.96957 |
| 10 | h2o.gbm | deskewed | eta=0.1;trees=894,depth=4 | 0.969 |
| 11 | nnet | deskew | M=100,maxit=250,decay=0.5 | 0.96029 |
| 12 | nnet | none | M=100,maxit=250,decay=0.5 | 0.94386 |

Table 10: Kaggle Results

# References


[1] 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.

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

[3] 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.

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

[5] 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.

[6] Gaëlle Loosli, Léon Bottou, and Patrice Simard. The infinite MNIST dataset. http://leon.bottou.org/projects/infimnist, 2007.

[7] 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.

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

[9] Rob Zinkov. Vowpal wabbit tutorial for the uninitiated. http://zinkov.com/posts/2013-08-13-vowpal-tutorial/, August 2013.

[10] 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.