Comparison of Analytical Techniques for the MNIST handwritten digits

George Fisher *

July 9, 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 [LeCun et al., 1998b], [Wikipedia, 2015]. 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 [LeCun et al., 1998a], [Cireşan et al., 2012]. Kaggle [Kaggle, 2015b] 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 A and B.

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 [Beissinger, 2015] and [Rosebrock, 2014] for instructions on setting up a GPU server for deep learning; see [Zwitch, 2013] for instructions for setting up an iPython server.

Programming Environment R version 3.2.0 [R Core Team, 2015], Python 2.7.9, iPython 3.1.0 [Pérez and Granger, 2007], 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` [Fisher, 2015].

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 [Hastie et al., 2011] 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 [Dean and Ghemawat, 2004] 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 [OpenCV,] which I implemented using an iPython (jupyter) notebook to deskew the entire database. See Appendix A.

Reading a large csv file can take a long time, the `fread` function of the `data.table` package in R is far superior to the standard `read.csv`; in fact, `data.table`s in general outperform `data.frame`s and all other data formats I have seen benchmarked; `pandas` seems to outperform `numpy.ndarray`s.

3 Single-hidden-layer neural networks

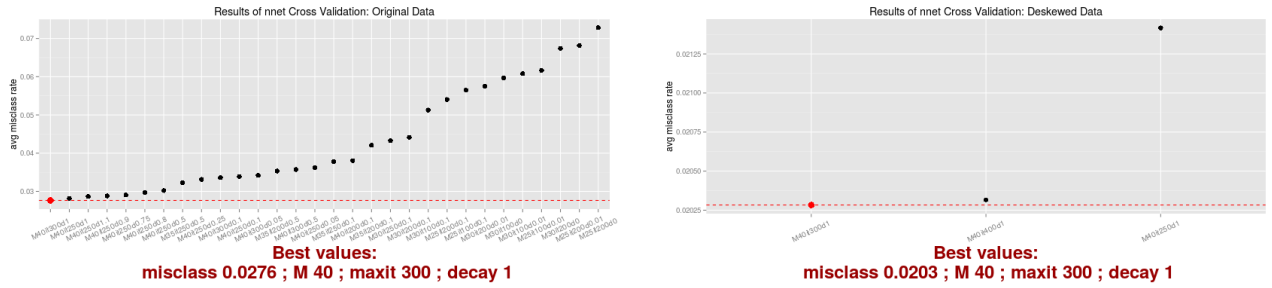


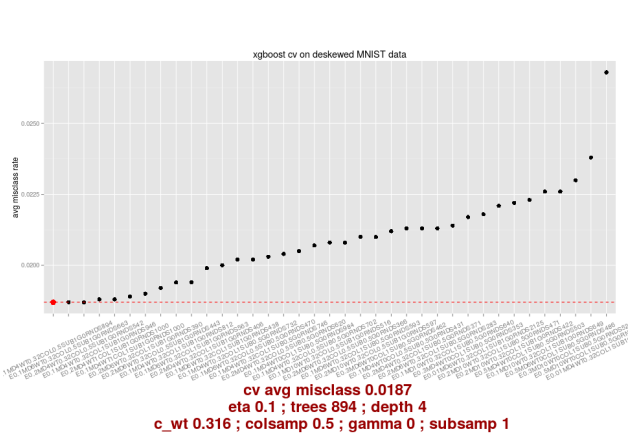
Figure 1: `nnet` 5-fold Cross Validation

	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

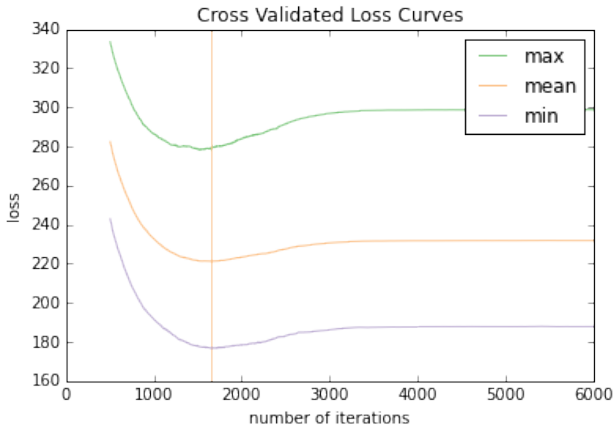
Table 1: Model full training set, predict test set

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 Elastic Net

6 Partial Least Squares

7 Vowpal Wabbit

8 K-Nearest Neighbors

9 Support Vector Machines

	package	language	preprocessing	misclass	benchmark	parameters
1	OpenCV SVM	Python	deskew, HOG	0.0560	0.0140	

Table 3: Model full training set, predict test set

10 Multi-Layer Neural Networks

11 Ensembles

12 Summary

	package	language	preprocessing	misclass	benchmark	parameters
1	avNNet	R	deskew	0.0126	0.0840	M=100,maxit=250,decay=1.0,repeats=3
2	gbm	H2O	deskewed	0.0159	0.0126	eta=0.1;trees=894,depth=4
3	nnet	R	none	0.0167	0.1200	M=100,maxit=250,decay=0.5
4	xgboost	R	deskewed	0.0171	0.0126	eta=0.1;trees=894,depth=4
5	gbm	scikit-learn	deskewed	0.0260	0.0126	eta=0.01;trees=1650,depth=4
6	OpenCV SVM	Python	deskew, HOG	0.0560	0.0140	

Table 4: Model full training set, predict test set

13 Kaggle Contest

The Digit Recognizer 'Getting Started' contest [Kaggle, 2015a] 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	xgboost	deskewed	eta=0.1;trees=894,depth=4	0.98014
2	avNNet	deskew	M=100,maxit=250,decay=1,repeats=3	0.97129
3	scikit.gbm	deskewed	eta=0.01;trees=1650,depth=4	0.96957
4	h2o.gbm	deskewed	eta=0.1;trees=894,depth=4	0.969
5	nnet	deskew	M=100,maxit=250,decay=0.5	0.96029
6	nnet	none	M=100,maxit=250,decay=0.5	0.94386

Table 5: Kaggle Results

A MNIST in Python: OpenCV Deskew

deskew

June 11, 2015

1 Deskew the MNIST training and test images

This set of scripts serves several purposes:
it provides two functions - a function to read the binary MNIST data into numpy.ndarray - a function to deskew the data * it deskews the data and displays the results * it saves the original and deskewed data plus labels as csv files

```
In [1]: %matplotlib inline
        import numpy as np
        import matplotlib.pyplot as plt
        import cv2
        print cv2.__version__
```

```
3.0.0-dev
```

2 OpenCV deskew function

```
In [2]: SZ = 28 # images are SZ x SZ grayscale

        affine_flags = cv2.WARP_INVERSE_MAP|cv2.INTER_LINEAR

        def deskew(img):
            m = cv2.moments(img)
            if abs(m['mu02']) < 1e-2:
                return img.copy()
            skew = m['mu11']/m['mu02']
            M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
            img = cv2.warpAffine(img,M,(SZ, SZ),flags=affine_flags)
            return img
```

3 Read MNIST binary-file data and convert to numpy.ndarray

thanks to http://g.sweyla.com/blog/2012/mnist-numpy/

```
In [3]: import os, struct
        from array import array as pyarray
        from numpy import append, array, int8, uint8, zeros

        def load_mnist(dataset="training", digits=None, path=None, asbytes=False, selection=None, return
            """
            Loads MNIST files into a 3D numpy array.
```

```
You have to download the data separately from [MNIST]_. It is recommended
to set the environment variable ``MNIST`` to point to the folder where you
put the data, so that you don't have to select path. On a Linux+bash setup,
this is done by adding the following to your ``.bashrc``::

    export MNIST=/path/to/mnist

Parameters
----------
dataset : str
    Either "training" or "testing", depending on which dataset you want to
    load.
digits : list
    Integer list of digits to load. The entire database is loaded if set to
    ``None``. Default is ``None``.
path : str
    Path to your MNIST datafiles. The default is ``None``, which will try
    to take the path from your environment variable ``MNIST``. The data can
    be downloaded from http://yann.lecun.com/exdb/mnist/.
asbytes : bool
    If True, returns data as ``numpy.uint8`` in [0, 255] as opposed to
    ``numpy.float64`` in [0.0, 1.0].
selection : slice
    Using a `slice` object, specify what subset of the dataset to load. An
    example is ``slice(0, 20, 2)``, which would load every other digit
    until--but not including--the twentieth.
return_labels : bool
    Specify whether or not labels should be returned. This is also a speed
    performance if digits are not specified, since then the labels file
    does not need to be read at all.
return_indicies : bool
    Specify whether or not to return the MNIST indices that were fetched.
    This is valuable only if digits is specified, because in that case it
    can be valuable to know how far
    in the database it reached.

Returns
-------
images : ndarray
    Image data of shape ``(N, rows, cols)``, where ``N`` is the number of images. If neithe
labels : ndarray
    Array of size ``N`` describing the labels. Returned only if ``return_labels`` is `True`
indices : ndarray
    The indices in the database that were returned.

Examples
--------
Assuming that you have downloaded the MNIST database and set the
environment variable ``£MNIST`` point to the folder, this will load all
images and labels from the training set:

>>> images, labels = ag.io.load_mnist('training') # doctest: +SKIP

Load 100 sevens from the testing set:
```

```
>>> sevens = ag.io.load_mnist('testing', digits=[7], selection=slice(0, 100), return_labels

"""

# The files are assumed to have these names and should be found in 'path'
files = {
    'training': ('train-images.idx3-ubyte', 'train-labels.idx1-ubyte'),
    'testing': ('t10k-images.idx3-ubyte', 't10k-labels.idx1-ubyte'),
}

if path is None:
    try:
        path = os.environ['MNIST']
    except KeyError:
        raise ValueError("Unspecified path requires environment variable $MNIST to be set")

try:
    images_fname = os.path.join(path, files[dataset][0])
    labels_fname = os.path.join(path, files[dataset][1])
except KeyError:
    raise ValueError("Data set must be 'testing' or 'training'")

# We can skip the labels file only if digits aren't specified and labels aren't asked for
if return_labels or digits is not None:
    flbl = open(labels_fname, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    labels_raw = pyarray("b", flbl.read())
    flbl.close()

fimg = open(images_fname, 'rb')
magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
images_raw = pyarray("B", fimg.read())
fimg.close()

if digits:
    indices = [k for k in range(size) if labels_raw[k] in digits]
else:
    indices = range(size)

if selection:
    indices = indices[selection]
N = len(indices)

images = zeros((N, rows, cols), dtype=uint8)

if return_labels:
    labels = zeros((N), dtype=int8)
for i, index in enumerate(indices):
    images[i] = array(images_raw[ indices[i]*rows*cols : (indices[i]+1)*rows*cols ]).reshape
    if return_labels:
        labels[i] = labels_raw[indices[i]]

if not asbytes:
```

```
            images = images.astype(float)/255.0

        ret = (images,)
        if return_labels:
            ret += (labels,)
        if return_indices:
            ret += (indices,)
        if len(ret) == 1:
            return ret[0] # Don't return a tuple of one
        else:
            return ret
```

4 Training Data

4.1 Read in the training images and labels

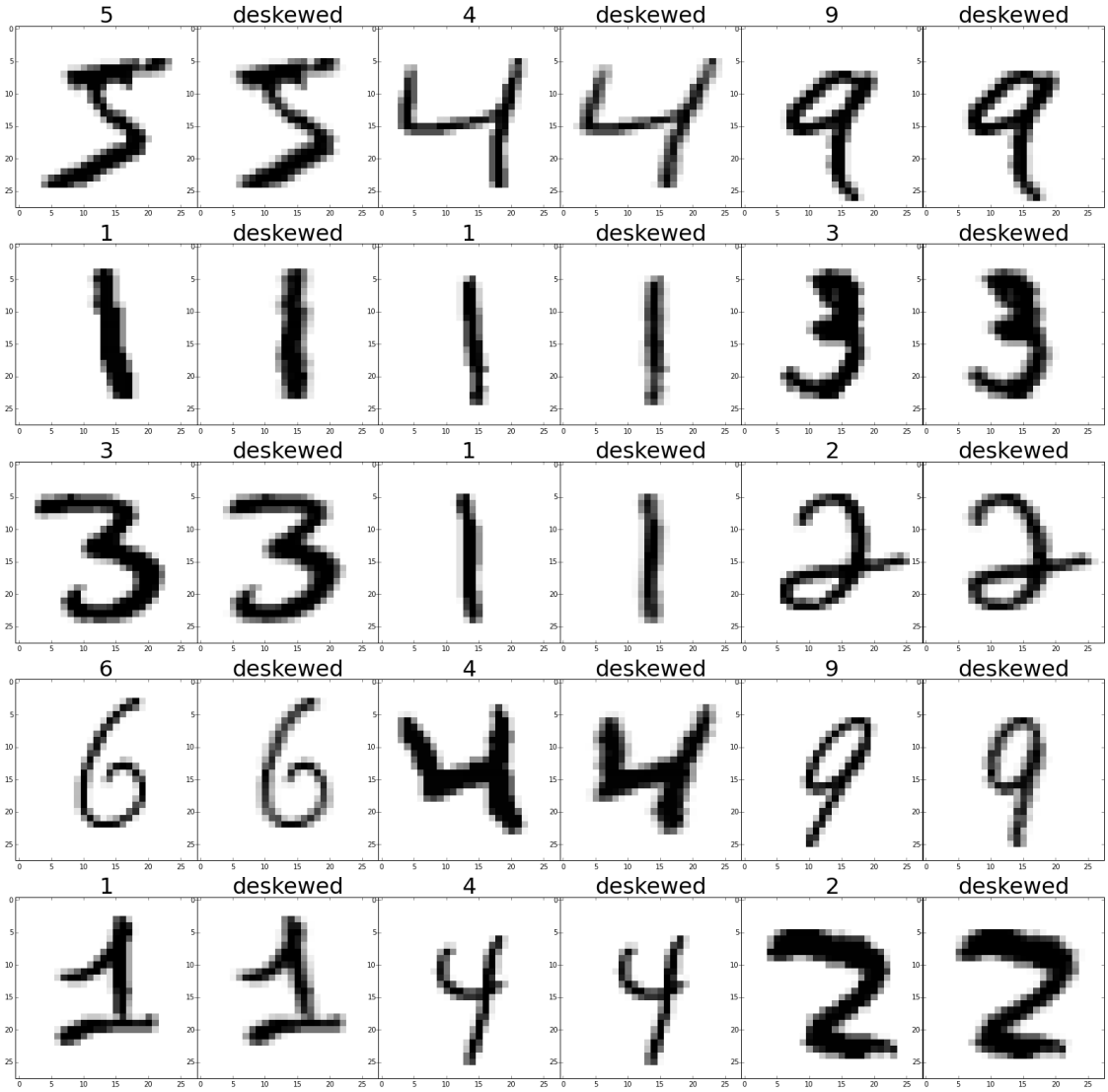
```
In [4]: images, labels = load_mnist('training', path="/home/george/Dropbox/MNIST/data")
```

4.2 Deskew training data

```
In [5]: images_deskewed = np.empty(np.shape(images))
        i = 0

        for img in images:
            images_deskewed[i] = deskew(np.reshape(img,(28,28)))
            i+=1
```

4.3 Show a sampling of training images before and after deskewing

```
In [6]: fig, axs = plt.subplots(5,6, figsize=(28,28), facecolor='w', edgecolor='k')
        fig.subplots_adjust(hspace = 0.0001, wspace=.001)

        axs = axs.ravel()

        for i in range(0,30,2):

            # convert from 1 x 784 to 28 x 28
            img     = np.reshape(images[i,:],(28,28))
            img_dsk = np.reshape(images_deskewed[i,:],(28,28))

            axs[i].imshow(img,cmap=plt.cm.gray_r, interpolation='nearest')
            axs[i].set_title(labels[i],fontsize=32)

            axs[i+1].imshow(img_dsk,cmap=plt.cm.gray_r, interpolation='nearest')
            axs[i+1].set_title('deskewed',fontsize=32)

        plt.show()
```



4.4 Save training data to csv

```
In [7]: import csv
        with open('data/train-images.csv', 'wb') as f:
            csv.writer(f).writerows([img.flatten() for img in images])
        with open('data/train-labels.csv', 'wb') as f:
            csv.writer(f).writerows([label.flatten() for label in labels])

        with open('data/train-images_deskewed.csv', 'wb') as f:
            csv.writer(f).writerows([img.flatten() for img in images_deskewed])
```

5 Test Data

```
In [8]: images = None
        labels = None
```

```
images_deskewed = None
```

5.1 Read in the test images and labels

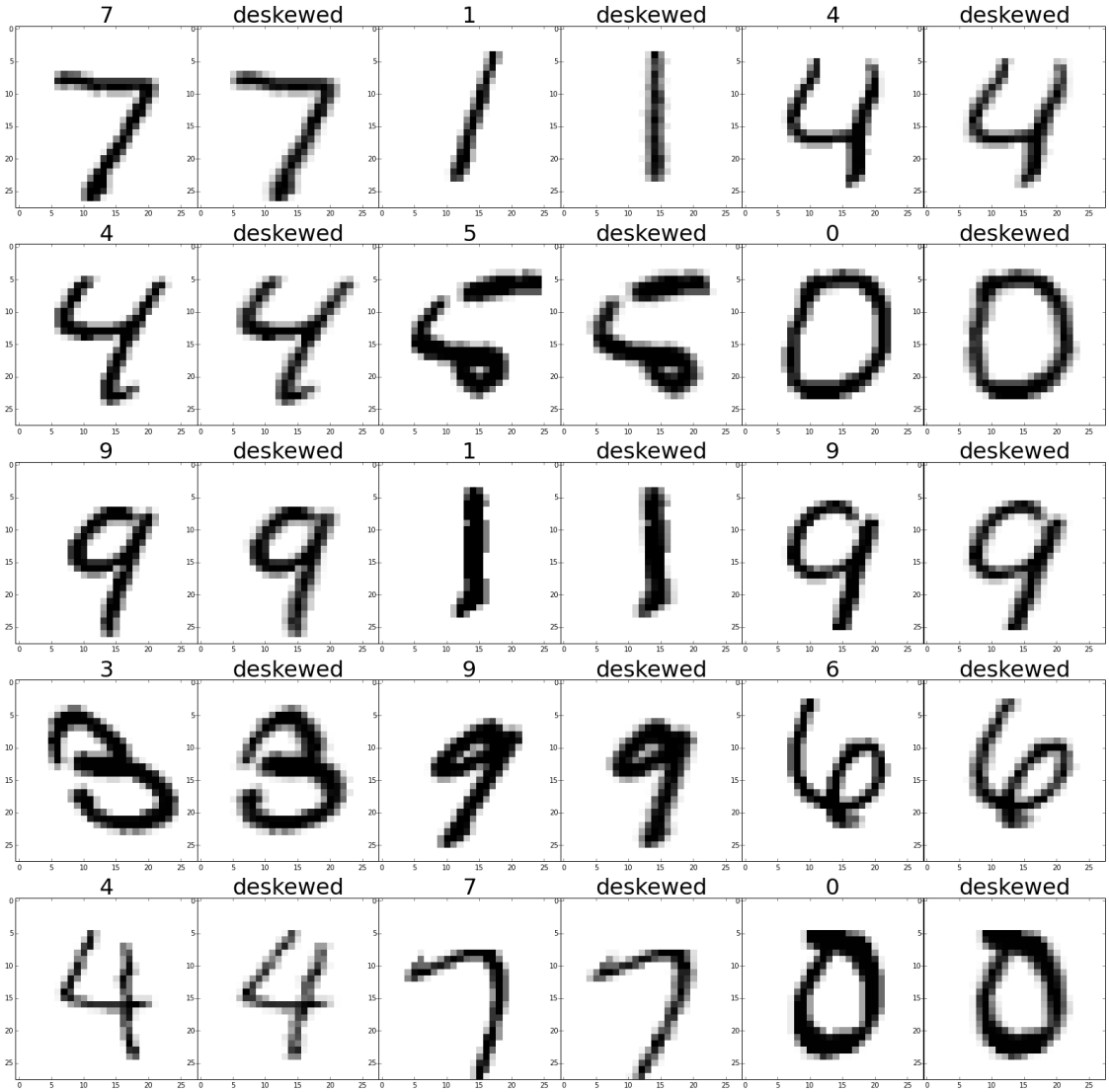
```
In [9]: images, labels = load_mnist('testing', path="/home/george/Dropbox/MNIST/data")
```

5.2 Deskew test data

```
In [10]: images_deskewed = np.empty(np.shape(images))
         i = 0

         for img in images:
             images_deskewed[i] = deskew(np.reshape(img,(28,28)))
             i+=1
```

5.3 Show a sampling of test images before and after deskewing

```
In [11]: fig, axs = plt.subplots(5,6, figsize=(28,28), facecolor='w', edgecolor='k')
         fig.subplots_adjust(hspace = 0.0001, wspace=.001)

         axs = axs.ravel()

         for i in range(0,30,2):

             # convert from 1 x 784 to 28 x 28
             img     = np.reshape(images[i,:],(28,28))
             img_dsk = np.reshape(images_deskewed[i,:],(28,28))

             axs[i].imshow(img,cmap=plt.cm.gray_r, interpolation='nearest')
             axs[i].set_title(labels[i],fontsize=32)

             axs[i+1].imshow(img_dsk,cmap=plt.cm.gray_r, interpolation='nearest')
             axs[i+1].set_title('deskewed',fontsize=32)

         plt.show()
```



5.4 Save test data to csv

```
In [12]: import csv
         with open('data/t10k-images.csv', 'wb') as f:
             csv.writer(f).writerows([img.flatten() for img in images])
         with open('data/t10k-labels.csv', 'wb') as f:
             csv.writer(f).writerows([label.flatten() for label in labels])

         with open('data/t10k-images_deskewed.csv', 'wb') as f:
             csv.writer(f).writerows([img.flatten() for img in images_deskewed])
```

B MNIST in R

```r
# see https://gist.github.com/brendano/39760

# Load the MNIST digit recognition dataset into R
# http://yann.lecun.com/exdb/mnist/
# assume you have all 4 files and gunzip'd them
# creates train$n, train$x, train$y  and test$n, test$x, test$y
# e.g. train$x is a 60000 x 784 matrix, each row is one digit (28x28)
# call:  show_digit(train$x[5,])   to see a digit.
# brendan o'connor - gist.github.com/39760 - anyall.org

load_mnist <- function() {
    load_image_file <- function(filename) {
        ret = list()
        f = file(filename,'rb')
        readBin(f,'integer',n=1,size=4,endian='big')
        ret$n = readBin(f,'integer',n=1,size=4,endian='big')
        nrow = readBin(f,'integer',n=1,size=4,endian='big')
        ncol = readBin(f,'integer',n=1,size=4,endian='big')
        x = readBin(f,'integer',n=ret$n*nrow*ncol,size=1,signed=F)
        ret$x = matrix(x, ncol=nrow*ncol, byrow=T)
        close(f)
        ret
    }
    load_label_file <- function(filename) {
        f = file(filename,'rb')
        readBin(f,'integer',n=1,size=4,endian='big')
        n = readBin(f,'integer',n=1,size=4,endian='big')
        y = readBin(f,'integer',n=n,size=1,signed=F)
        close(f)
        y
    }
    train <<- load_image_file('data/train-images.idx3-ubyte')
    test <<- load_image_file('data/t10k-images.idx3-ubyte')

    train$y <<- load_label_file('data/train-labels.idx1-ubyte')
    test$y <<- load_label_file('data/t10k-labels.idx1-ubyte')
}


show_digit <- function(arr784, col=gray(12:1/12), ...) {
    image(matrix(arr784, nrow=28)[,28:1], col=col, ...)
}

print_16 = function(starting_at=1, X=trainX, Y=trainY) {
    # print a 4x4 of images in the training set
    # starting at index=starting_at
    opar = par(no.readonly=TRUE)
    par(mfrow=c(4,4))
    for (i in seq(from=starting_at, length.out=16)){
        show_digit(matrix(as.numeric(X[i,]),28,28),
                   main=Y[i],
                   xlab=paste("index",i))
    }
    par(opar)
}
```

References

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

[Cireşan et al., 2012] Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. Computer Vision and Pattern Recognition. CVPR 2012, p. 3642-3649.

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters . http://research.google.com/archive/mapreduce.html.

[Fisher, 2015] Fisher, G. (2015). MNIST: Initial setup plus nnet R algorithm.

[Hastie et al., 2011] Hastie, T., Tibshirani, R., and Friedman, J. (2011). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer.

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

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

[LeCun et al., 1998a] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998a). Gradient-based learning applied to document recognition. *Proceedings - IEEE*.

[LeCun et al., 1998b] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998b). THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/. accessed June 2015.

[OpenCV,] 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.

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

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

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

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

[Zwitch, 2013] Zwitch, R. (2013). 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/.