MNIST CLASSIFICATION: NNS vs KNN (prelim)

The MNIST database of handwritten digits, available from this page[1] , has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

R routines provided at the end of document for downloading and processing the MNIST dataset.

RESULTS

At every step, NNS is significantly more accurate than KNN for k=1 and k=5.[2] We present the R command in red and the output of total errors in blue.

```
> require(devtools); install_github('OVVO-Financial/NNS', ref = "NNS-Beta-Version")
> require(NNS)
```

TEST 1: 10% Train on 10% Test

NNS: 90% accuracy

```
> sum(pmin(1,abs(round(NNS.reg(train$x[1:6000,],train$y[1:6000],point.est =
test$x[1:1000,],n.best=1,dist = "L1",plot=FALSE,order='max')$Point.est)-test$y[1:1000])))
[1] 100
```

KNN 1:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:6000,], test = test$x[1:1000,],cl =
train$y[1:6000], k=1)))-test$y[1:1000])))
[1] 474
```

KNN 5:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:6000,], test = test$x[1:1000,],cl =
train$y[1:6000], k=5)))-test$y[1:1000])))
[1] 285
```

[1] http://yann.lecun.com/exdb/mnist/

[2] k=5 suggestion: http://acgrama.blogspot.com/2012/09/knn-with-euclidean-distance-on-mnist.html

TEST 2: 20% Train on 20% Test

NNS: 91.9% accuracy

```
> sum(pmin(1,abs(round(NNS.reg(train$x[1:12000,],train$y[1:12000],point.est =
test$x[1:2000,],n.best=1,dist = "L1",plot=FALSE)$Point.est)-test$y[1:2000])))
[1] 163
```

KNN 1:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:12000,], test = test$x[1:2000,],cl =
train$y[1:12000], k=1)))-test$y[1:2000])))
[1] 888
```

KNN 5:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:12000,], test = test$x[1:2000,],cl =
train$y[1:12000], k=5)))-test$y[1:2000])))
[1] 639
```

TEST 3: 30% Train on 100 Test Observations

NNS: Only 1 wrong!

```
> sum(pmin(1,abs(round(NNS.reg(train$x[1:18000,],train$y[1:18000],point.est =
test$x[1:100,],n.best=1,dist = "L1",plot=FALSE,order='max')$Point.est)-test$y[1:100])))
[1] 1
```

KNN 1:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:100,],cl =
train$y[1:18000], k=1)))-test$y[1:100])))
[1] 39
```

KNN 5:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:100,],cl =
train$y[1:18000], k=5)))-test$y[1:100])))
[1] 26
```

TEST 4: 30% Train on 18% Test

NNS: 92.67% accuracy

```
> sum(pmin(1,abs(round(NNS.reg(train$x[1:18000,],train$y[1:18000],point.est =
test$x[1:1800,],n.best=1,dist = "L1",plot=FALSE,order='max')$Point.est)-test$y[1:1800])))
[1] 132
```

KNN 1:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:1800,],cl =
train$y[1:18000], k=1)))-test$y[1:1800])))
[1] 763
```

KNN 5:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:1800,],cl =
train$y[1:18000], k=5)))-test$y[1:1800])))
[1] 528
```

TEST 5: 30% Train on 30% Test

NNS: 92.4% accuracy

```
> sum(pmin(1,abs(round(NNS.reg(train$x[1:18000,],train$y[1:18000],point.est =
test$x[1:3000,],n.best=1,dist = "L1",plot=FALSE,order='max')$Point.est)-test$y[1:3000])))
[1] 228
```

KNN 1:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:3000,],cl =
train$y[1:18000], k=1)))-test$y[1:3000])))
[1] 1257
```

KNN 5:

```
> sum(pmin(1,abs(as.numeric(as.character(knn(train = train$x[1:18000,], test = test$x[1:3000,],cl =
train$y[1:18000], k=5)))-test$y[1:3000])))
[1] 883
```

SUMMARY

NNS is consistently more accurate than KNN. Current KNN rates for the full MNIST dataset are > 95% accurate[3] suggesting NNS in the upper 98-99% range based on the NNS errors vs KNN errors.

Memory and time constraints prohibited the full dataset analysis. Given NNS' superior performance on this benchmark problem as well as demonstrated excellence in other classification and regression problems[4], a concentrated effort to optimize NNS' code for memory and processing constraints seems more than warranted. An optimized NNS would offer a viable robust alternative to neural networks currently employed in machine learning.

---

[3] http://yann.lecun.com/exdb/mnist/

[4] See the following:

Classification Using NNS Clustering Analysis https://ssrn.com/abstract=2864711

Clustering and Curve Fitting by Line Segments https://ssrn.com/abstract=2861339

R Routines

DOWNLOAD MNIST DATA ROUTINE
https://gist.github.com/johnbaums/882ad1e458e13b96a3d1

```
get.mnist <- function(dir=NULL) {
  # dir: the path containing the extracted files:
  #  train-images-idx3-ubyte ;  train-labels-idx1-ubyte
  #  t10k-images-idx3-ubyte ;  t10k-labels-idx1-ubyte
  if(is.null(dir)) {
    require(R.utils)
    dir <- getwd()
    u <- c('http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz',
           'http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz',
           'http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz',
           'http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz')
    sapply(u, function(x) {
      download.file(x, f <- file.path(dir, basename(x)))
      gunzip(f) })
  }
 }
```

LOAD MNIST DATA ROUTINE
https://gist.github.com/primaryobjects/b0c8333834debbc15be4

```
library(caret)
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('train-images-idx3-ubyte')
  test <<- load_image_file('t10k-images-idx3-ubyte')
```

```
  train$y <<- load_label_file('train-labels-idx1-ubyte')
  test$y <<- load_label_file('t10k-labels-idx1-ubyte')
}
train <- data.frame()
test <- data.frame()
# Load data.
load_mnist()
# Normalize pixel intensity 255 greyscale.
train$x <- train$x / 255
```