MNIST Example

Carlos Carvalho, Mladen Kolar, Rob McCulloch

10/21/2015

In this example, we will explore the famous MNIST handwritten digits data set.

Data are provided by Yann LeCun and can be downloaded here: http://yann.lecun.com/exdb/mnist/index.html.

For convenience, the data can be also downloaded from a GitHub repository. For example, you can clone the repository using the following command:

> git clone https://github.com/ChicagoBoothML/DATA___LeCun___MNISTDigits.git

Remark: Data can also be downloaded from Kaggle: https://www.kaggle.com/c/digit-recognizer/data. Note that the data available from the Kaggle's website is not partitioned in the same way into training and test sets. Below, I will be using data from Yann LeCun's website.

The MNIST data set has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image.

Each observation is a grey-scale image sized 28 by 28 pixels. The columns are the pixel numbers, ranging from pixel 0 to pixel 783 (784 total pixels), which have elements taking values from 0 to 255 (white is 0 and 255 is black). Thus, our observations each have 784 feature values.

The goal is to build a model that will be presented with an image of a numerical digit (0-9) and the model must predict which digit is being shown.

Data Import

Let us load the data.

You will need to change the code below to match the directory where you downloaded MNIST files.

```
# MNIST files in Git repository
# MNIST_DIR = "/home/mkolar/projects/mlRepos/DATA___LeCun___MNISTDigits"

# MNIST files in the current directory
MNIST_DIR = "."
```

We will also need special code to load the data set, since the data set is stored in the IDX file format. You can find more about the file format here.

```
load_mnist <- function(folder) {
  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')
```

```r
    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(file.path(folder, 'train-images.idx3-ubyte'))
  test <- load_image_file(file.path(folder, 't10k-images.idx3-ubyte'))

  train$y <- load_label_file(file.path(folder, 'train-labels.idx1-ubyte'))
  test$y <- load_label_file(file.path(folder, 't10k-labels.idx1-ubyte'))

  list(train=train, test=test)
}
```

Using the above code, we can load the digits

```r
digit.data = load_mnist(MNIST_DIR)
```

The training sample size

```r
digit.data$train$n
```

```
## [1] 60000
```

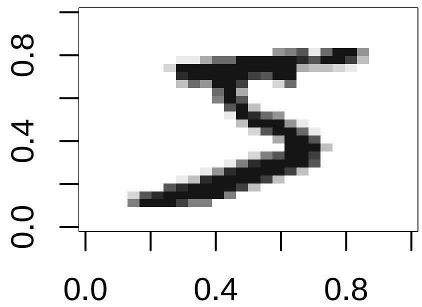
Number of features

```r
ncol(digit.data$train$x)
```

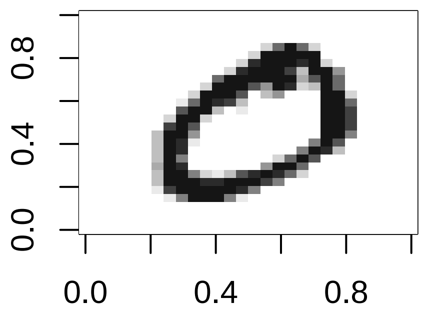
```
## [1] 784
```

Each image is a row of the training matrix. The following code will represent one image.

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

show_digit(digit.data$train$x[1, ])
```



```r
show_digit(digit.data$train$x[2, ])
```



Pixels are organized into images like this:

```
001 002 003 ... 026 027 028
029 030 031 ... 054 055 056
057 058 059 ... 082 083 084
 |   |   |  ...  |   |   |
729 730 731 ... 754 755 756
757 758 759 ... 782 783 784
```

Logistic regression

```r
library(glmnet)
```

```
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-2
```

```r
if (file.exists("glmnet_mnist.RData")) {
  load("glmnet_mnist.RData")
} else {
  glm_fit = glmnet(x=digit.data$train$x, y=as.factor(digit.data$train$y), family="multinomial",
                   type.logistic="modified.Newton")
  save(glm_fit, file = "glmnet_mnist.RData")
}

phat = predict(glm_fit, digit.data$test$x, s=1.464817e-03, type = "response")
yhat = apply(phat,1,which.max)
ot = table(yhat, digit.data$test$y)
sum(diag(ot)) / 10000 # accuracy
```

```
## [1] 0.9161
```

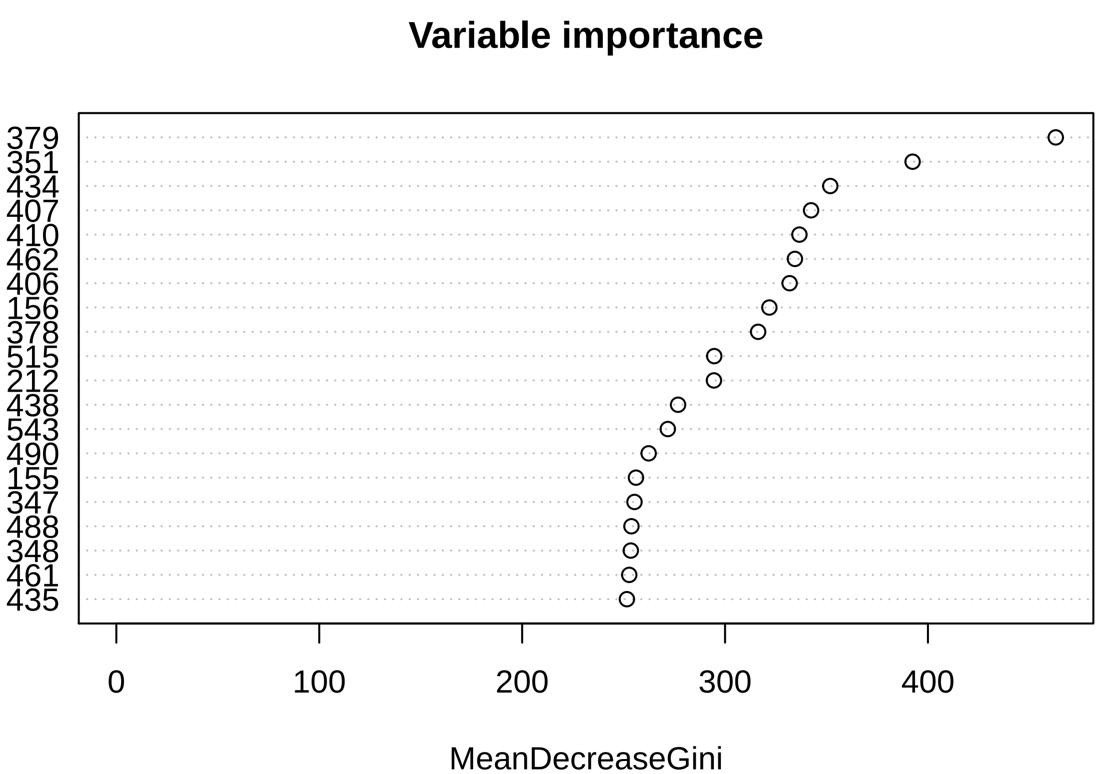
Random Forests

```r
train = digit.data$train$x
test = digit.data$test$x
label = as.factor(digit.data$train$y)

if (file.exists("rf_mtry_28_MNIST.RData")) {
  load("rf_mtry_28_MNIST.RData")
} else {
  num_trees = 1000

  rf_28 = randomForest(
    x=train,
    y=label,
    xtests=test,
    sampsize=6000,   # sample about 10% of data
    ntree=num_trees,
    mtry=28,         # try 28 = sqrt(784) features at each split
    importance=TRUE,
    nodesize=100     # need this many observations in the leaf
  )

  save(rf_28,file = "rf_mtry_28_MNIST.RData")
}
rf_28
```

```
##
## Call:
##  randomForest(x = train, y = label, ntree = num_trees, mtry = 28,      nodesize = 100, importance = 
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 28
##
##         OOB estimate of  error rate: 5.29%
## Confusion matrix:
##      0    1    2    3    4    5    6    7    8    9 class.error
## 0 5790    1   10    1    5    7   20    2   83    4  0.02245484
## 1    0 6559   63   27   15   15   10   15   31    7  0.02714328
## 2   39    9 5657   40   57    3   34   44   60   15  0.05052031
## 3   17   16  125 5648   12   89   15   59  101   49  0.07877997
## 4   15    7   21    1 5548    0   43   12   35  160  0.05032523
## 5   43   16   15   83   14 5073   63    9   69   36  0.06419480
## 6   36   14    7    0   16   62 5740    0   43    0  0.03007773
## 7    8   28   94    9   56    1    0 5895   33  141  0.05905826
## 8   17   45   43   72   37   49   31    8 5440  109  0.07024440
## 9   37   14   33   97  105   23    4   78   80 5478  0.07917297
```

Important pixels

```r
varImpPlot(rf_28, type=2, n.var=20, main="Variable importance")
```



Confusion matrix for the test set

```r
predicted.test = predict(rf_28, test)

confusionMatrix(table(predicted.test,digit.data$test$y))
```

```
## Confusion Matrix and Statistics
##
##
## predicted.test    0    1    2    3    4    5    6    7    8    9
##              0  967    0    5    2    2    6    9    1    5    8
##              1    0 1118    0    0    1    3    3    4    1    6
##              2    0    3  978   23    2    1    1   27    7    2
##              3    0    4    9  942    0   18    0    5    7   14
##              4    0    0   11    2  929    5    6    4    6   18
##              5    3    1    1   13    1  835    7    0    5    1
##              6    4    4    7    0    6    9  925    0    9    1
##              7    1    1    9   11    0    3    0  960    4    4
##              8    5    4   10   13    7    8    7    5  912   16
##              9    0    0    2    4   34    4    0   22   18  939
##
## Overall Statistics
##
##                Accuracy : 0.9505
##                  95% CI : (0.9461, 0.9547)
##     No Information Rate : 0.1135
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.945
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9867   0.9850   0.9477   0.9327   0.9460   0.9361
## Specificity            0.9958   0.9980   0.9926   0.9937   0.9942   0.9965
## Pos Pred Value         0.9622   0.9842   0.9368   0.9429   0.9470   0.9631
## Neg Pred Value         0.9986   0.9981   0.9940   0.9924   0.9941   0.9938
## Prevalence             0.0980   0.1135   0.1032   0.1010   0.0982   0.0892
## Detection Rate         0.0967   0.1118   0.0978   0.0942   0.0929   0.0835
## Detection Prevalence   0.1005   0.1136   0.1044   0.0999   0.0981   0.0867
## Balanced Accuracy      0.9913   0.9915   0.9702   0.9632   0.9701   0.9663
##                      Class: 6 Class: 7 Class: 8 Class: 9
## Sensitivity            0.9656   0.9339   0.9363   0.9306
## Specificity            0.9956   0.9963   0.9917   0.9907
## Pos Pred Value         0.9585   0.9668   0.9240   0.9179
## Neg Pred Value         0.9963   0.9925   0.9931   0.9922
## Prevalence             0.0958   0.1028   0.0974   0.1009
## Detection Rate         0.0925   0.0960   0.0912   0.0939
## Detection Prevalence   0.0965   0.0993   0.0987   0.1023
## Balanced Accuracy      0.9806   0.9651   0.9640   0.9606
```