R Notebook: features for classification

Contents

```r
library(devtools);
```

```
## Loading required package: usethis
```

```r
library(humanVerseWSU);

path.github = "https://raw.githubusercontent.com/MonteShaffer/humanVerseWSU/master/";

include.me = paste0(path.github, "misc/functions-mnist.R");
source_url( include.me );
```

```
## SHA-1 hash of file is 834b34ab7745de4b56c99ba99ce527e381b45699
```

```
## Linking to ImageMagick 6.9.11.32
## Enabled features: cairo, fontconfig, freetype, lcms, pango, rsvg, webp
## Disabled features: fftw, ghostscript, x11
```

```r
include.me = paste0(path.github, "humanVerseWSU/R/functions-encryption.R");
source_url( include.me );
```

```
## SHA-1 hash of file is da71dde620bed33db055778b752eefb476f7bf6b
```

```r
path.to.nascent = "/Users/jaileefoster/Desktop/stat419/_git_/WSU_STATS419_FALL2020/nascent/";

folder.mnist = "mnist-png/";
path.to.mnist = paste0(path.to.nascent, folder.mnist);
```

```
training.files = mnist.grabFiles("training",
                        path.to.mnist, folder.mnist);
```

MNIST classification (cont'd)

We see an image , what can we do with it? That is, what features can we extract from the data provided? Last time, we introduced the "eigenvalue" as a potential domain of features. Now, we will introduce additional features to make the classification structure more robust.

NOTE: Recall, eigen requires square matrices.

Decision-making outline

Data analytics is about making decisions, and lots of them. When we build a classification structure, we need to try and organize the decisions as independent and cascading.

Training Data

Sometimes the number "6" and "9" have a line under it. I didn't find samples with this case.

From those, we will try to "test" some eigenvector approaches using the first "12" images found in the data set.

Data inputs

- Original data is in PNG format, 28x28
- We could store the scaled form on an interval [0,1] or we could just assign a value of 0 or 1 to each entry in the matrix.
- We could possibly divide the matrix into sub-matrices for more resolution in feature extraction. We could add rows/colums of zeroes to get to 30x30 and then we could do "trecile" divisions, 9 sub-grids each at 10x10.
- We could try and center the image within the available space (center both horizontally and vertically)

Feature extraction

- Eigen of the entire matrix, Eigen.half where we truncate the dimensions of the data-reduced vector.
- rowSums/colSums of the entire matrix as a representation of its xy-profiling.
- Similar features (eigen, eigen.half, rowsums, colsums) for a sub-divided matrix. 28 x 28 can already be divided in halves (4 subgrids) or fourths (16 subgrids). If we augmented the matrix to 30 x 30, we can divide by 3. There is a reason 12 is called a "perfect number" - if we had a multiple of twelve, we could divide by half, third, fourth, and sixth. Could we trim the images down to 24x24? The number "seven: from before suggests that may be possible, so let's review our data of the"training set".

Can we truncate to 24x24?

```
n = length(training.files);
m = length(training.files[[1]]); # we assume balanced panel
total = n * m;

info.cols = c("number.letter","n","m", "rows.with.data", "cols.with.data");

result = as.data.frame( matrix(0,
              nrow=total, ncol= length(info.cols) ) );
  colnames(result) = info.cols;
```

```r
result$number.letter = "";

i = 1;
for(d in 1:n) # true digit
    {
    num = d; if(num == 10) { num = 0; }
    for(r in 1:m) # replicate
      {
      img.file = training.files[[d]][r];
      img.matrix = mnist.grabMatrixFromImage(img.file,
                                            path.to.nascent);
      img.matrix.rs = rowSums(img.matrix);
      img.matrix.cs = colSums(img.matrix);

      result[i,]$number.letter = paste0(num,"-",letters[r]);
      result[i,]$n = d;
      result[i,]$m = r;
      result[i,]$rows.with.data = sum(img.matrix.rs > 0);
      result[i,]$cols.with.data = sum(img.matrix.cs > 0);
      i = 1 + i;
      }
    }

result;
```

```
##    number.letter  n m rows.with.data cols.with.data
## 1            1-a  1 1             20              8
## 2            1-b  1 2             20             16
## 3            1-c  1 3             20              9
## 4            1-d  1 4             20             10
## 5            2-a  2 1             18             20
## 6            2-b  2 2             20             20
## 7            2-c  2 3             18             20
## 8            2-d  2 4             18             20
## 9            3-a  3 1             16             20
## 10           3-b  3 2             20             14
## 11           3-c  3 3             20             20
## 12           3-d  3 4             20             18
## 13           4-a  4 1             20             14
## 14           4-b  4 2             20             20
## 15           4-c  4 3             20             16
## 16           4-d  4 4             20             18
## 17           5-a  5 1             20             17
## 18           5-b  5 2             20             20
## 19           5-c  5 3             20             12
## 20           5-d  5 4             20             18
## 21           6-a  6 1             20             14
## 22           6-b  6 2             20             17
## 23           6-c  6 3             20             14
## 24           6-d  6 4             20             14
## 25           7-a  7 1             20             17
## 26           7-b  7 2             16             20
## 27           7-c  7 3             20             12
## 28          7-d  7 4              18             20
## 29          8-a  8 1              20             12
## 30          8-b  8 2              20             14
## 31          8-c  8 3              20             17
## 32          8-d  8 4              20             10
## 33          9-a  9 1              20             13
## 34          9-b  9 2              20             14
## 35          9-c  9 3              20             10
## 36          9-d  9 4              20             18
## 37          0-a 10 1              20             20
## 38          0-b 10 2              20             12
## 39          0-c 10 3              20             18
## 40          0-d 10 4              20             18
```

```
max(result$rows.with.data);
```

```
## [1] 20
```

```
max(result$cols.with.data);
```

```
## [1] 20
```

It looks like the answer is yes (the max dimensions of our “training data” is 20 x 20). We will try and center the data with a default “top-left” bias (if odd/even prevent perfect centering).

We could justify in a form other than “center” but it would not matter much with the exception of the number “one”.

```
img.file = training.files[[2]][3];  ### maybe rotate "8"s?
img.matrix = mnist.grabMatrixFromImage(img.file,
                              path.to.nascent);
#im.printImageMatrix(img.matrix);
dim(img.matrix);
```

```
## [1] 28 28
```

```
img.matrix.t = mnist.trimMatrix(img.matrix);
im.printImageMatrix(img.matrix.t);
```

```
##       [,1]
##  [1,] "00000111111100000000"
##  [2,] "00011111111100000000"
##  [3,] "00011110011110000000"
##  [4,] "00111000001110000000"
##  [5,] "00111000001111000000"
##  [6,] "00110000001111000000"
##  [7,] "00000000001111000000"
##  [8,] "00000000000111000000"
##  [9,] "00000000001111000000"
## [10,] "00000000001111001111"
## [11,] "00000000111111111111"
## [12,] "00111111111111111110"
## [13,] "11111111111111000000"
## [14,] "11111111011110000000"
## [15,] "11100001111100000000"
## [16,] "11100011111000000000"
## [17,] "11111111110000000000"
## [18,] "01111111000000000000"
```

```r
dim(img.matrix.t);
```

```
## [1] 18 20
```

```r
img.matrix.c = mnist.centerMatrix(img.matrix, c(24,24)); # this may truncate "as-is"
im.printImageMatrix(img.matrix.c, 3); # split into sub-grids
```

```
##       [,1]
##  [1,] "00000000|00000000|00000000"
##  [2,] "00000000|00000000|00000000"
##  [3,] "00000000|00000000|00000000"
##  [4,] "00000001|11111100|00000000"
##  [5,] "00000111|11111100|00000000"
##  [6,] "00000111|10011110|00000000"
##  [7,] "00001110|00001110|00000000"
##  [8,] "00001110|00001111|00000000"
##  [9,] "--------+--------+--------"
## [10,] "00001100|00001111|00000000"
## [11,] "00000000|00001111|00000000"
## [12,] "00000000|00000111|00000000"
## [13,] "00000000|00001111|00000000"
## [14,] "00000000|00001111|00111100"
## [15,] "00000000|00111111|11111100"
## [16,] "00001111|11111111|11111000"
## [17,] "00111111|11111111|00000000"
## [18,] "--------+--------+--------"
## [19,] "00111111|11011110|00000000"
## [20,] "00111000|01111100|00000000"
## [21,] "00111000|11111000|00000000"
## [22,] "00111111|11110000|00000000"
## [23,] "00011111|11000000|00000000"
## [24,] "00000000|00000000|00000000"
## [25,] "00000000|00000000|00000000"
## [26,] "00000000|00000000|00000000"
```

```r
im.printImageMatrix(img.matrix.c, 6);
```

```
##       [,1]
##  [1,] "0000|0000|0000|0000|0000|0000"
##  [2,] "0000|0000|0000|0000|0000|0000"
##  [3,] "0000|0000|0000|0000|0000|0000"
##  [4,] "0000|0001|1111|1100|0000|0000"
##  [5,] "----+----+----+----+----+----"
##  [6,] "0000|0111|1111|1100|0000|0000"
##  [7,] "0000|0111|1001|1110|0000|0000"
##  [8,] "0000|1110|0000|1110|0000|0000"
##  [9,] "0000|1110|0000|1111|0000|0000"
## [10,] "----+----+----+----+----+----"
## [11,] "0000|1100|0000|1111|0000|0000"
## [12,] "0000|0000|0000|1111|0000|0000"
## [13,] "0000|0000|0000|0111|0000|0000"
## [14,] "0000|0000|0000|1111|0000|0000"
## [15,] "----+----+----+----+----+----"
## [16,] "0000|0000|0000|1111|0011|1100"
## [17,] "0000|0000|0011|1111|1111|1100"
## [18,] "0000|1111|1111|1111|1111|1000"
## [19,] "0011|1111|1111|1111|0000|0000"
## [20,] "----+----+----+----+----+----"
## [21,] "0011|1111|1101|1110|0000|0000"
## [22,] "0011|1000|0111|1100|0000|0000"
## [23,] "0011|1000|1111|1000|0000|0000"
## [24,] "0011|1111|1111|0000|0000|0000"
## [25,] "----+----+----+----+----+----"
## [26,] "0001|1111|1100|0000|0000|0000"
## [27,] "0000|0000|0000|0000|0000|0000"
## [28,] "0000|0000|0000|0000|0000|0000"
## [29,] "0000|0000|0000|0000|0000|0000"
```

```r
im.printImageMatrix(img.matrix.c, 6, raw=TRUE);
```

```
##       [,1]
##  [1,] "0000|0000|0000|0000|0000|0000"
##  [2,] "0000|0000|0000|0000|0000|0000"
##  [3,] "0000|0000|0000|0000|0000|0000"
##  [4,] "0000|0004|68XX|8100|0000|0000"
##  [5,] "----+----+----+----+----+----"
##  [6,] "0000|017X|977X|X800|0000|0000"
##  [7,] "0000|0895|1002|9X50|0000|0000"
##  [8,] "0000|3X70|0000|5X90|0000|0000"
##  [9,] "0000|3930|0000|2X91|0000|0000"
## [10,] "----+----+----+----+----+----"
## [11,] "0000|0000|0000|2XX4|0000|0000"
## [12,] "0000|0000|0000|08X6|0000|0000"
## [13,] "0000|0000|0000|08X6|0000|0000"
## [14,] "0000|0000|0000|08X5|0000|0000"
## [15,] "----+----+----+----+----+----"
## [16,] "0000|0000|0000|2XX2|0013|5200"
## [17,] "0000|0000|0001|4XX8|689X|9600"
## [18,] "0000|0135|888X|XXXX|9952|1000"
## [19,] "0005|8XXX|X99X|XX62|0000|0000"
## [20,] "----+----+----+----+----+----"
## [21,] "007X|X763|2108|X900|0000|0000"
## [22,] "00X9|2000|005X|9200|0000|0000"
## [23,] "00X8|0000|07XX|3000|0000|0000"
## [24,] "008X|6754|8X82|0000|0000|0000"
## [25,] "----+----+----+----+----+----"
## [26,] "0005|XXXX|7300|0000|0000|0000"
## [27,] "0000|0000|0000|0000|0000|0000"
## [28,] "0000|0000|0000|0000|0000|0000"
## [29,] "0000|0000|0000|0000|0000|0000"
```

```r
im.printImageMatrix(img.matrix.c, 6, raw=TRUE, raw.r="I");
```

```
##       [,1]
##  [1,] "0000|0000|0000|0000|0000|0000"
##  [2,] "0000|0000|0000|0000|0000|0000"
##  [3,] "0000|0000|0000|0000|0000|0000"
##  [4,] "0000|0004|68II|8100|0000|0000"
##  [5,] "----+----+----+----+----+----"
##  [6,] "0000|017I|977I|I800|0000|0000"
##  [7,] "0000|0895|1002|9I50|0000|0000"
##  [8,] "0000|3I70|0000|5I90|0000|0000"
##  [9,] "0000|3930|0000|2I91|0000|0000"
## [10,] "----+----+----+----+----+----"
## [11,] "0000|0000|0000|2II4|0000|0000"
## [12,] "0000|0000|0000|08I6|0000|0000"
## [13,] "0000|0000|0000|08I6|0000|0000"
## [14,] "0000|0000|0000|08I5|0000|0000"
## [15,] "----+----+----+----+----+----"
## [16,] "0000|0000|0000|2II2|0013|5200"
## [17,] "0000|0000|0001|4II8|689I|9600"
## [18,] "0000|0135|888I|IIII|9952|1000"
## [19,] "0005|8III|I99I|II62|0000|0000"
## [20,] "----+----+----+----+----+----"
## [21,] "007I|I763|2108|I900|0000|0000"
## [22,] "00I9|2000|005I|9200|0000|0000"
## [23,] "00I8|0000|07II|3000|0000|0000"
## [24,] "008I|6754|8I82|0000|0000|0000"
## [25,] "----+----+----+----+----+----"
## [26,] "0005|IIII|7300|0000|0000|0000"
## [27,] "0000|0000|0000|0000|0000|0000"
## [28,] "0000|0000|0000|0000|0000|0000"
## [29,] "0000|0000|0000|0000|0000|0000"
```

```
  # numbers are 0.1 = 1; 0.2 = 2; ... 1.0 = "X"
dim(img.matrix.c);
```

```
## [1] 24 24
```

```
mine.sweep = mnist.countMineSweepMoves(img.matrix.c);
mine.sweep$count;
```

```
## [1] 3
```

```
im.printImageMatrix(mine.sweep$matrix, 3);
```

```
##       [,1]
##  [1,] "........|........|........"
##  [2,] "........|........|........"
##  [3,] "........|........|........"
##  [4,] ".......1|111111..|........"
##  [5,] ".....111|111111..|........"
##  [6,] ".....111|1..1111.|........"
##  [7,] "....111.|....111.|........"
##  [8,] "....111.|....1111|........"
##  [9,] "--------+--------+--------"
## [10,] "....11..|....1111|........"
## [11,] "........|....1111|........"
## [12,] "........|.....111|........"
## [13,] "........|....1111|........"
## [14,] "........|....1111|..1111.."
## [15,] "........|..111111|111111.."
## [16,] "....1111|11111111|11111..."
## [17,] "..111111|11111111|........"
## [18,] "--------+--------+--------"
## [19,] "..111111|11.1111.|........"
## [20,] "..111...|.11111..|........"
## [21,] "..111...|11111...|........"
## [22,] "..111111|1111....|........"
## [23,] "...11111|11......|........"
## [24,] "........|........|........"
## [25,] "........|........|........"
## [26,] "........|........|........"
```

```
## odd tests ... [[1]][3] is 20 x 9
# img.matrix.t = img.matrix.t[1:19,1:9] # ... now 19 x 9
# mnist.zeroFillMatrix(img.matrix.t, c(24,24));
```

Decision-making conclusions

We now want to take our exploration and build logical cascades. Ideally, these are independent, so we could possibly examine the performance of all possible feature-structure permutations.

- Size of matrix: if we don't resize to 24x24 we cannot easily do both a 2- and 3- split partition. **matrix.size:** "raw", "24x24" …
- Data in matrix: binary (either zero or one) or scaled on continuum of [0,1]. **data.form:** "b", "c" for binary, continuous
- Divisions in matrix: 1 (no division), 2 (cut into 4 sub-grids), 3, (cut into 9 sub-grids). A 2-row, 3-col cut may be very informative, but the result would be a non-square matrix. **matrix.division:** 1, 2, 3 (4, 6 are possible as well with 24x24 … this depends on the size of the matrix, if not conformable, we will skip the option). Let's say we develop a naming terminology that the top-left is the first sub-grid, and we read them from left-to-right, top-to-bottom similar to how we read text. We can then stack the sub-grids into one long vector. Recall an eigen was of length 28 originally, a rowsum would be of that same length. If we cut it into 4 sub-grids, we would have 4 sub-eigens of length 12 or a total of 48 in length. If we cut it into 9 sub-grids, we would have 9 sub-eigens of length 8 or a total of 72 in length.
- Features from matrix: eigenvalue, eigenvalue.half, rowsums, colsums, and others (??? can you think of others). This choice is non-exculsive, we can do them all. **matrix.features:**
- Comparison of features: we are using cosine similarity (and considered euclidean distance). Again we could do others. **comparison.methods:**
- Statistic of comparison: we can do top-N, or some pooling across variants with mean and median. **decision.stats:** For now, we will do a statistical comparison independently. Later, we can figure out a way to sum various methods and what weightings to provide to the sums. This is the idea of an ensemble. https://en.wikipedia.org/wiki/Ensemble_learning

We have an updated game plan. So we need to have functions that keep these tasks as independent as possible. We will apply the same functions to the "training" data and "testing" data and then will apply the comparison functions between each testing data record.

Options for Functions

```
# we could have the options in nested keys or a paste0 flat key.  Either way, we will have some for loo

option = list();
option$matrix.size        = c("raw", "24x24");
  #option$matrix.size = c("24x24");
option$data.form          = c("b", "c");   # c("binary", "continuous");
  #option$data.form = c("c");
option$matrix.division    = c(1,2,3,4,6);
  #option$matrix.division = c(3);
# this option is currently hardcoded ...
option$matrix.features    = c("eigen", "eigen.half", "eigen.fourth", "eigen.third", "eigen.sixth",  "gr
  #option$matrix.features = c("gridcount");
option$comparison.methods = c("sim.cosine","dist.euclidean");
option$decision.stats     = c("mean", "median", "sd", "median+sd", "top^1", "top^2", "top^3", "top^4",
# option$decision.stats = c("top^1");

training.data = mnist.prepareTrainingData(training.files,
                     path.to.mnist, folder.mnist, option);
```

```
## [1] "function mnist.prepareTrainingData with cache.file: training-3d90d656675340cf036f0e5b99afd3bf.r
## [1] "time elapsed: 0.1 secs"
```

Testing Data

Some Logical Hypotheses It is important as you delve into your data to try and understand the data and how the various options may influence the results. Here are a few of my hypotheses:

- A division of the matrix into 3 cuts will be superior to an even number of cuts. If you manually look at the data, it will create differences in 9 grids that are distinct. This creates variance.
- A `top^1` will outperform a `top^2` which will outperform a `top^3`, and so on.

[What are some conjectures you have about the data?]

– WRITE SOMETHING HERE –

```

# Cntrl + Alt + I ... # Thanks, Nathan
testing.files = mnist.grabFiles("testing",
                        path.to.mnist, folder.mnist);

# example of same preparation for a single "comparison" element
test.one = mnist.prepareOneTest(testing.files,
                  d=1, r=1,
                  path.to.mnist, folder.mnist, option);




result.one = mnist.performOneTest(testing.files, training.data,
                  d=2, r=2,
                  path.to.mnist, folder.mnist, option);

best.of.best =  whichMaxFreq(result.one$best);
best.of.best;
```

(One) Calculate & Summarize

```
## [1] 0
```

```
outcome.details = as.numeric(unlist(result.one[,6:length(result.one)]));

table(outcome.details);
```

```
## outcome.details
##    0    1    2    3    4    5    6    7    8    9
## 2592  204 1152  504   54  610  126    8   96   30
```

```
whichMaxFreq(outcome.details);
```

```
## [1] 0
```

At the moment, the testing approaches are remaining "independent". From above, you can see that the goal would be to minimize "bad answers" by either changing the options or developing a "dependent" algorithm: use this information from this option, and this information from this other option, to create an ideal outcome.

With `d=2; r=2;` I have the correct answer coming in second place behind the number "0" … so how can I distinguish between a "2" and a "0" better?

– WRITE SOMETHING HERE –

Note: With the above, we can peak inside the approach. Just change `d` and `r` from the testing group.

```
# this will take awhile... we could divide the tests up into children processes and use multi-threads .
## took about 300 seconds on my computer with "all options"
## so we cache it ...
final = mnist.performAllTest(testing.files, training.data,
                  path.to.mnist, folder.mnist, option);
```

(ALL) Calculate

```
## [1] "function mnist.performAllTest with cache.file: testing-a791a82550062bc370b4d6d54ec8bf9f.rds"
## [1] "time elapsed: 0.1 secs"
```

```
# str(final); # we have the raw dataframe panel, true, and best.of.best
```

```
final.summary = mnist.resultSummary(final, option);

final.summary;
```

(ALL) Summarize

```
## $s.raw
## [1] 0.1736979
##
## $s.24x24
## [1] 0.198125
##
## $b
## [1] 0.1889323
##
## $c
## [1] 0.1889974
##
## $n1
## [1] 0.1946615
##
## $n2
## [1] 0.1829427
##
## $n3
## [1] 0.2171875
##
## $n4
## [1] 0.1576823
##
## $n6
## [1] 0.2239583
##
## $eigen
## [1] 0.1864583
##
## $eigen.half
## [1] 0.1888021
##
## $eigen.fourth
## [1] 0.1888021
##
## $eigen.third
## [1] 0.1927083
##
## $eigen.sixth
## [1] 0.1885417
##
## $gridcount
## [1] 0.1854167
##
## $rowsums
## [1] 0.1755208
##
## $colsums
## [1] 0.2054688
##
## $sim.cosine
## [1] 0.1901693
##
## $dist.euclidean
## [1] 0.1877604
##
## $stats.mean
## [1] 0.3070312
##
## $stats.median
## [1] 0.2742188
##
## $stats.sd
## [1] 0.1234375
##
## $`stats.median+sd`
## [1] 0.2609375
##
## $`stats.top^1`
## [1] 0.3617187
##
## $`stats.top^2`
## [1] 0.2960937
##
## $`stats.top^3`
## [1] 0.3101563
##
## $`stats.top^4`
## [1] 0.2773438
##
## $`stats.top^5`
## [1] 0.2835937
##
## $`stats.top^1+mean+median+sd`
## [1] 0.3445313
##
## $`stats.top^1+mean+median`
## [1] 0.3578125
##
## $`stats.top^1+mean`
## [1] 0.3671875
##
## $`stats.top^1+median`
## [1] 0.3640625
##
## $`stats.top^2+mean+median+sd`
## [1] 0.3265625
##
## $`stats.top^2+mean+median`
## [1] 0.334375
##
## $`stats.top^2+mean`
## [1] 0.3351562
##
## $`stats.top^2+median`
## [1] 0.325
##
## $`stats.top^3+mean+median+sd`
## [1] 0.325
##
## $`stats.top^3+mean+median`
## [1] 0.3265625
##
## $`stats.top^3+mean`
## [1] 0.334375
##
## $`stats.top^3+median`
## [1] 0.3195312
```

```r
## matrix.size
sub = wildcardSearch( paste0("24x24",".*"), "data.id", df=final$data);
## data.form
sub = wildcardSearch( paste0("*.","c",".*"), "data.id", df=sub);
## matrix.division
sub = wildcardSearch( paste0("*.",3,"_*"), "data.id", df=sub);
## matrix.features
sub = wildcardSearch( paste0("*_","eigen.third"), "data.id", df=sub);
## comparison.methods
sub = subsetDataFrame( sub, "comparison.method", "==" , "sim.cosine");
## decision.stats
result = list();
```

```r
pick.one = NULL;
for(decision.stats in option$decision.stats)
    {
    key.name = paste0("stats.",decision.stats);
    name.idx = which(names(final$data) == key.name );
    stats1 = sub[,c(1:2,name.idx)];
    #test1 = sum(sub$true == sub$best) / nrow(sub);
    test1 = sum(stats1[,1] == stats1[,3]) / nrow(stats1); # compare to individual test...
    if(key.name == "stats.top^2+mean") { pick.one=stats1; }
    result[[key.name]] = test1;
    }
result;
```

(Instructor) Dependent Example

```
## $stats.mean
## [1] 0.4
##
## $stats.median
## [1] 0.4
##
## $stats.sd
## [1] 0.3
##
## $`stats.median+sd`
## [1] 0.2
##
## $`stats.top^1`
## [1] 0.4
##
## $`stats.top^2`
## [1] 0.4
##
## $`stats.top^3`
## [1] 0.5
##
## $`stats.top^4`
## [1] 0.5
##
## $`stats.top^5`
## [1] 0.5
##
## $`stats.top^1+mean+median+sd`
## [1] 0.4
##
## $`stats.top^1+mean+median`
## [1] 0.4
##
## $`stats.top^1+mean`
## [1] 0.4
##
## $`stats.top^1+median`
## [1] 0.4
##
```

```
## $`stats.top^2+mean+median+sd`
## [1] 0.5
##
## $`stats.top^2+mean+median`
## [1] 0.5
##
## $`stats.top^2+mean`
## [1] 0.5
##
## $`stats.top^2+median`
## [1] 0.5
##
## $`stats.top^3+mean+median+sd`
## [1] 0.5
##
## $`stats.top^3+mean+median`
## [1] 0.5
##
## $`stats.top^3+mean`
## [1] 0.5
##
## $`stats.top^3+median`
## [1] 0.5
```

```
pick.one;
```

```
##       true best stats.top^2+mean
## 215      1    1                1
## 471      1    1                1
## 727      1    1                1
## 983      1    1                1
## 1239     1    1                1
## 1495     1    1                1
## 1751     1    1                1
## 2007     1    1                1
## 2263     1    1                1
## 2519     1    1                1
## 2775     1    1                1
## 3031     1    1                1
## 3287     2    0                0
## 3543     2    0                0
## 3799     2    0                0
## 4055     2    0                0
## 4311     2    0                0
## 4567     2    0                0
## 4823     2    0                0
## 5079     2    0                0
## 5335     2    0                0
## 5591     2    0                0
## 5847     2    0                0
## 6103     2    0                0
## 6359     3    1                5
## 6615     3    1                5
## 6871     3    1                5
## 7127     3    2                5
## 7383     3    2                5
## 7639     3    3                5
## 7895     3    4                5
## 8151     3    2                5
## 8407     3    5                5
## 8663     3    5                5
## 8919     3    5                5
## 9175     3    2                5
## 9431     4    0                0
## 9687     4    0                0
## 9943     4    0                0
## 10199    4    0                0
## 10455    4    0                0
## 10711    4    0                0
## 10967    4    0                0
## 11223    4    0                0
## 11479    4    0                0
## 11735    4    0                0
## 11991    4    0                0
## 12247    4    0                0
## 12503    5    2                5
## 12759    5    2                5
## 13015    5    3                5
## 13271    5    3                5
## 13527    5    5                5
## 13783    5    2                5
## 14039    5    3                5
## 14295    5    2                5
## 14551    5    5                5
## 14807    5    1                5
## 15063    5    4                5
## 15319    5    5                5
## 15575    6    6                6
## 15831    6    4                6
## 16087    6    3                6
## 16343    6    5                6
## 16599    6    1                6
## 16855    6    2                6
## 17111    6    5                6
## 17367    6    2                6
## 17623    6    6                6
## 17879    6    6                6
## 18135    6    2                6
## 18391    6    3                6
## 18647    7    0                0
## 18903    7    0                0
## 19159    7    0                0
## 19415    7    0                0
## 19671    7    0                0
## 19927    7    0                0
## 20183    7    0                0
## 20439    7    0                0
## 20695    7    0                0
## 20951    7    0                0
## 21207     7    0                0
## 21463     7    0                0
## 21719     8    7                8
## 21975     8    3                8
## 22231     8    3                8
## 22487     8    8                8
## 22743     8    5                8
## 22999     8    7                8
## 23255     8    8                8
## 23511     8    6                8
## 23767     8    6                8
## 24023     8    6                8
## 24279     8    2                8
## 24535     8    1                8
## 24791     9    1                1
## 25047     9    1                1
## 25303     9    1                1
## 25559     9    1                1
## 25815     9    1                1
## 26071     9    1                1
## 26327     9    1                1
## 26583     9    1                1
## 26839     9    1                1
## 27095     9    1                1
## 27351     9    1                1
## 27607     9    1                1
## 27863     0    0                0
## 28119     0    0                0
## 28375     0    0                0
## 28631     0    0                0
## 28887     0    0                0
## 29143     0    0                0
## 29399     0    0                0
## 29655     0    0                0
## 29911     0    0                0
## 30167     0    0                0
## 30423     0    0                0
## 30679     0    0                0
```

For a given `key.name` you can print the `stats1` and see the result. Recall that "pure-chance" is 1/10, so we are now doing much better than the first notebook. Whether there is a bug in the code, or we need to think more carefully about a given feature, we can still improve on this framework.

The number "0" is causing some issues. How could we adjust our setup to not let the "0" bully the results. In the `stats1` I printed, we see that "2", "4", "7" are losing to "0" big time. If you dig into a single comparison-decision, the correct answer is often the "next-highest" result behind the "0". If we understand xy-profiling ("colsums" and "rowsums"), it does make sense, the "0" has a similar profile. I also observe that "3" is getting mapped to "5" and again is the "next-highest" result. And the "9" is getting mapped to a "1". That one I cannot fully comprehend as to why.

In summary, we can be optimistic, the "3", "5", "6", "8" should be the only noisy ones and an "ensemble" can likely take care of them. We have to improve a few things, but we can get there.

(Student) Dependent Example Can you beat `0.50` by selecting different subsets of the data? If so, what do you get? Why would you choose those options?

```
# clues are available in function mnist.resultSummary
```

– WRITE SOMETHING HERE –

Final Comments

Keeping track of permutations is very challenging. That is why data and logic organization are imperative. This is naturally an iterative process; however, the cookbook is basically the same for most classification problems:

- How we select the data (matrix.size and data.form)
- How we divide the data (matrix.division)
- What data features we can extract (matrix.features)
- How to compare the given features (comparison.methods)
- What statistics will determine the prediction (decision.stats)

Under the hood, you could examine the function `mnist.doComparisons` that would give you the raw data for each comparison. With this raw data, a library like "SuperLearner" would enable you to define custom learning functions and it would ascertain the best "ensemble" of features to use to get a better score. This is where computation benefits. There are too many permutations for a human mind to consider at once. These "ensemble" tools just do a bunch of for-loops and analyze a best set of linear combinations.

Netflix

- https://en.wikipedia.org/wiki/Netflix_Prize
- https://web.archive.org/web/20131206030554/http://www.netflixprize.com/leaderboard

There was a contest several years ago on improving the Netflix "suggestions" algorithm with very little data input. The winning team devised a way to "correlate" movies based on merely the words in the title. The also created an "ensemble" as you can see from their quote below:

"…, we list all 107 results that were blended to deliver RMSE=0.8712, with their weights within the ensemble. The results are grouped based on the type of method that produced them."

- https://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf

SuperLearner

At the end of the semester, I may make an (optional) notebook on "SuperLearner" that shows how and "ensemble" can identify the best combinations for this data. We likely have enough options.

Tune with small subset

We could certain include more training/testing data. However, it is best to "tune" your logic on a small dataset so you can understand what is going on. The full dataset has about 60,000 training data and over 10,000 testing data. The "law of large numbers" will help us, but at the appropriate time.

Custom Classifier

I intentionally chose to build a custom classifier based on "eigen". I also did so to demonstrate good processes in iterating through data setup, organization, and preparation. This process will enable you to build your own classifier, or merely plug in one of the many classifiers that are available: http://yann.lecun.com/exdb/mnist/

```
# a for-loop of this would tabulate the results over all test.data ...
# https://cran.r-project.org/web/packages/SuperLearner/vignettes/Guide-to-SuperLearner.html
# randomForest
```