Problem Set #6

Zejia Chen, Francis Lin, Xuanyang Lin

March 17th, 2022

```r
# loading the data
setwd("/Users/zejiachen/Desktop/Sspring 2022/Statstical Learning/DataAnalysis#2")
train_x <- fread("MNISTTrainX.csv")
train_y <- fread("MNISTTrainY.csv")
validate_x <- fread("MNISTValidationX.csv")
validate_y <- fread("MNISTValidationY.csv")
test_x <- fread("MNISTTestXRand.csv")
test_y <- fread("MNISTTestYRand.csv")
```

This is the first midterm data analysis exercise for QTM 385 - Introduction to Statistical Learning. This exercise will use a single data set to ask a few questions related to creating predictive models for a classification. This data set is larger and messier than those that have been given for the problem set exercises. It is intended to provide you with a "real-world" scenario you might see if attempting to build a predictive model for a question you care about. This assignment is worth 15% of your final grade and your final solutions (a .Rmd file, a corresponding rendered document, and 3 .csv files with your predictions) should be submitted by 11:59 PM on April 11th.

Unlike the problem sets, there is less guidance as to what methods to use and how to interpret results. The idea is that you can treat this like a real analysis exercise - applying different models, seeing what works and doesn't work, presenting the best possible model but being transparent about the downsides of your choice. For each of the three tasks, you should consider a variety of potential methods and choose a single one as your "best" model.

A review of classification methods that we've covered in this course:

1. Discriminative Methods

- Logistic Regression
- Multinomial Logistic Regression
- Shrinkage Approaches to Logistic Regression (LASSO and Ridge)
- Generalized Additive Logistic Regression

2. Generative Methods

- Bayes' Theorem for Discrete Features
- Linear and Quadratic Disciminant Analysis
- Naive Bayes classifiers (with a mixture of discrete and continuous margins $\pm$ kernel density estimates)

3. Geometric Methods

- Support Vector Classifiers
- Support Vector Machines with Kernelized Features (Polynomial and RBF)

4. Flexible Classifiers

- KNN and DANN
- Classification Trees (Bagged Trees, Random Forests, Probability Forests)
- Boosted Trees (AdaBoost via GBM, XGBoost)

- Stacked Classifiers

Data Set Overview

This assignment revolves around a standard in the classification literature - the MNIST handwritten digits database. The MNIST data set is a collection of 70,000 handwritten digits taken from handwritten zip codes on letters sent via the USPS. A single instance (or observation) is a 28×28 pixel image of a handwritten digit. Each of the 784 features associated with each image is a **color value** associated with a single pixel. Color has been coded on a grayscale with 0 corresponding to white (a "paper" pixel), 256 corresponding to black (a "pencil" pixel), and all values in between corresponding to varying levels of gray in accordance with the direction of the grayscale. In reality, all pixels are either paper or pencil, but various aliasing algorithms and rounding algorithms needed to handle pixels that take on both paper and pencil values lead some pixels to fall at a gray level.

To make analysis possible, the MNIST data set processes the images to ensure that the images are centered - the center of pencil mass is shifted to be around pixel 14,14 - and *deskewed* - the pencil pixels are shiften to be as vertical as possible. The original deskewing algorithm used for the MNIST data set was not very effective, though, so many images are still skewed in one direction or another.

Each image is associated with a label of 0 through 9 coded by a series of human coders. There has been significant quality control on this data set that ensures that all of the labels are correct!

MNIST is typically used to benchmark classification algorithms. The data set is of good size for many state-of-the-art classification methods and presents a somewhat difficult classification problem with 10 categories. The 10 class problem has been applied to many different algorithms to varying levels of success. The original website for MNIST from Yann LeCun, Corinna Cortes, and Christopher J.C. Burges contains error rates on a common test set of 10,000 instances for a number of different classification approaches. For example, over the 10 class problem using multinomial logistic regression (annoyingly referred to as a 1-layer neural network) has a 12% error rate on the test set. On the other hand, a committee of 35 convolutional neural nets was able to misclassify only 23 of the test digits!

We'll be using a modified version of this data set to run a series of classification models for different classification tasks. Your final task will be to build a classification model for the full 10 class problem!

Modifying the MNIST Data Set

The original data set contains 70,000 instances of labeled 28×28 pixels. Since we have neither the computational power nor the time to deal with data of this magnitude, I've made some convenience alterations to the data set. Here, I'll outline exactly what I did.

First, to reduce the number of pixels for each image, I reduced the resolution of the images from 784 pixels to 196 pixels. To do this, I created 196 2×2 pixel groups and averaged the color values across the four included pixels. As seen in the figure below, this drastically reduced the dimensionality of the predictor space at the cost of a little less clarity in the resulting pixel images.

Second, to further reduce the number of predictors, I removed full rows and columns of pixels that are part of the paper bounding box (e.g. all paper pixels) in more than 99% of images. This resulted in removing any pixel in columns 1 and 14 and in rows 1 and 14. This reduced the number of pixels in each image to 144.

Finally, I added just a little random noise to any paper pixels (pixels with colors values less than 5) to inject a little variance into any mostly paper coumns of pixels. This will matter very little for classification accuracy but will allow methods that require inversion of a covariance matrix (logistic regression, QDA, etc.) to run without any additional processing needed. These final two steps have little effect on the images compared to the 14×14 pixel images.

Working with the MNIST Data

Using the 12×12 images, I've created six data sets for you to work with:

1. `MNISTTrainX.csv` and `MNISTTrainY.csv` which contain 25,000 images or image labels. For each of the ten digits, there are 2,500 observations. I've split the images and the labels into separate files to make plotting the digit images as easy as possible.

2. `MNISTValidationX.csv` and `MNISTValidationY.csv` which contain 15,000 images or image labels. For each of the ten digits, there are 1,500 observations. These data sets can be used to measure out of sample predictive accuracy for the classification methods.

3. `MNISTTestXRand.csv` and `MNISTTestYRand.csv` which contain 10,000 images. There are no labels for this data set. These 10,000 observations (all ten handwritten digits are represented at least 200 times in this test set) will act as a hidden test set that will be used to measure the accuracy of your approaches on my held out test set. `MNISTTestYRand.csv` is an "empty" .csv (all the labels are zeros) that includes the image row corresponding to the test set and a column that can be used to store your final digit prediction for the corresponding row in `MNISTTestX.csv`. Your final deliverable will include three different sets of predictions for this data set.

Each of the data sets that include pixel values are organized in a consistent way. Each image/row is associated with 144 predictors - each of the 144 pixels in the image. The pixels are *vectorized* from their matrix form by row - the first feature is row 1 column 1, the second is row 1 column 2, the 12th feature is row 1 column 12, the 13th feature is row 2 column 1, the 25th features is row 3 column 1, etc. I've provided feature names in each data set to assist in this interpretation.

The reason that this organization is so important is that you will frequently want to convert between the vector of pixels and the actual 2D image (a matrix). We can easily convert the row of 144 pixels to a 12 by 12 matrix by passing the vector to a matrix **by row**. Let x be a vector, then this can be achieved easily in R by using `matrix(nrow = 12, ncol = 12, x, byrow = TRUE)`. This logic can be applied in any scenario where we receive a vector of quantities related to each pixel (coefficients from logistic regression, for example).

Since it is difficult to look at the numbers and know exactly what we're working with, we need a consistent plotting method. The easiest way to plot the MNIST data is to use the `image()` function in R. For your convenience, you can use the `plot_digit()` function below:

```
plot_digit <- function(x, bw = FALSE,...){
  if(sqrt(length(x)) != round(sqrt(length(x)))){
    stop(print("Not a square image! Something is wrong here."))
  }
  n <- sqrt(length(x))
  if(bw == TRUE){
    x <- as.numeric(x > 50)*256
  }
  par(pty = "s")
  image(matrix(as.matrix(x), nrow = n)[,n:1], col = gray(12:1 / 12), ...)
}
```

`plot_digit()` takes two argument: 1. `x` - a vector of squared integer length that is organized in row-column order (like our MNIST data) and 2. `bw` - a logical (TRUE/FALSE) that tells the function to plot the full grayscale **or** approximately round each pixel to either be white (paper) or black (pencil). `plot_digit` also accepts any arguments that are applicable to the base plot function in R: `main` to add a plot title, `xlab` or `ylab` to add x and y axis labels, etc. `image` in R is a legacy function from SPlus that has a quirk that it plots columns in reverse order. Be careful if using `image` to create your own figures! You can change the color scheme for `image` using different color range methods. Look online to find instructions for this. There are other methods of plotting the MNIST data in R and Python that can be easily found via a quick Google search.

Warning! A common mistake will be to pass a vector of length 145 (pixels + label) to this function. If you get an error, check and make sure that you aren't passing the label to the function!

Question 1 (15 pts.)

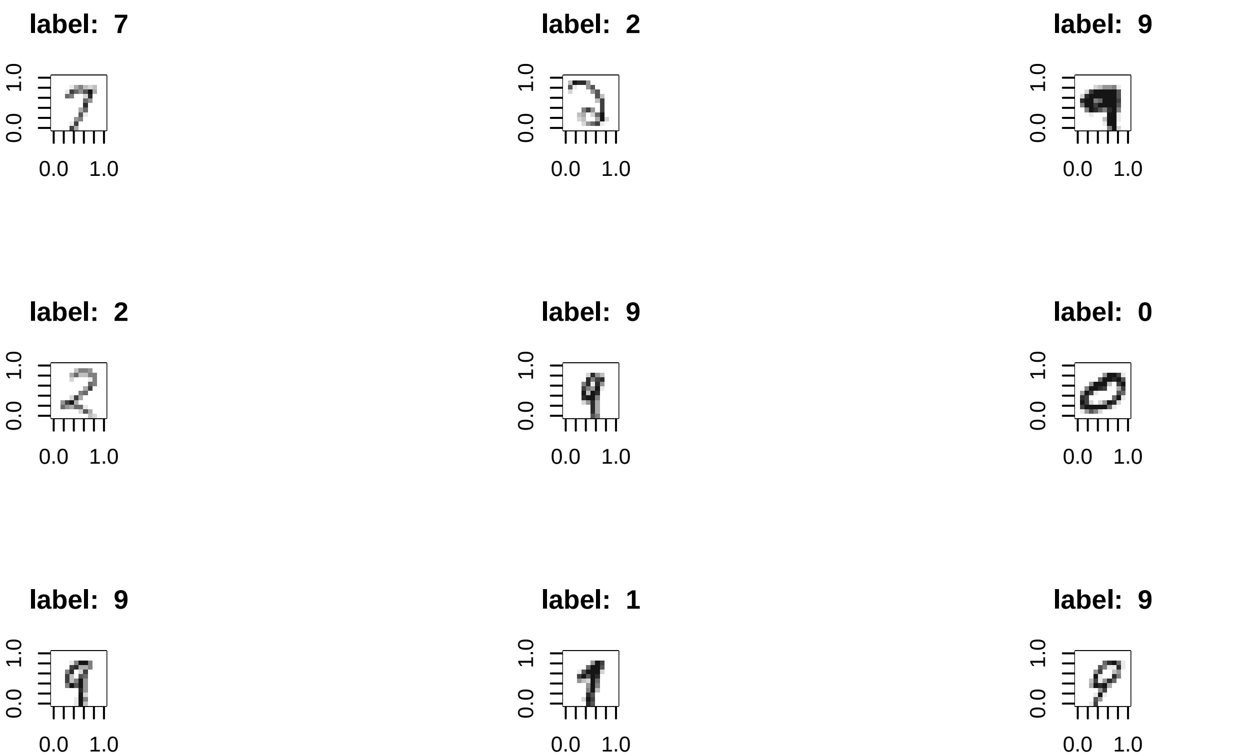
Let's start by working with the training data to gain some comfort working with the MNIST data.

Part 1 (5 pts.)

Create a plot that shows 9 random images in the training data. Label each image with its corresponding label. Do the images match the labels?

```
sample_9 <- sample(1:25000, 9)

par(mfrow = c(3,3))
for (i in sample_9){
  label = train_y[i, 1]
  plot_digit(train_x[i,], bw = FALSE, main = paste("label: ", label))
}
```




```
par(mfrow = c(1,1))
```

Part 2 (5 pts.)

For each digit (0 - 9), compute the average within class pixel value for each of the 144 pixels across the training data. Create a plot that shows the 10 average digits. Which class seems to show the most within class variation over images? Which class seems to the least within class variation over images?

```
meanPixel <- data.frame(matrix(nrow=10,ncol=144))
```

```
for (i in 0:10){
  index <- which(train_y$label == i)
  subsets <- as.matrix(train_x[index,])
  for (j in 1:144){
    meanPixel[i + 1,j] = mean(subsets[,j])
  }
}

head(meanPixel)
```

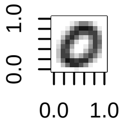
```
##          X1       X2       X3       X4        X5        X6        X7        X8
## 1 2.465105 2.522030 2.567673 2.589474  2.914744  3.816678  4.531078  4.508223
## 2 2.527473 2.489656 2.472274 2.556489  2.570255  2.854144  3.325927  2.950887
## 3 2.475565 2.574087 3.498163 7.367956 14.414192 21.817748 24.049480 19.087107
## 4 2.516169 2.467417 2.590944 2.966534  4.318283  5.824365  5.751006  4.445568
## 5 2.512653 2.553874 2.481628 2.507038  2.677434  2.724938  2.865923  3.016629
## 6 2.503118 2.494946 2.454187 2.512955  2.615625  3.147024  3.834242  4.033972
##          X9      X10      X11      X12      X13      X14       X15       X16
## 1  3.813080 3.055749 2.616443 2.519116 2.512549 2.535504  3.058565  7.855194
## 2  2.891412 2.625402 2.559470 2.476667 2.506196 2.497976  2.611132  3.064081
## 3 10.187743 4.484698 2.622012 2.549562 2.457359 4.117735 14.438655 41.614503
## 4  3.485089 2.926889 2.558739 2.486570 2.572041 5.043470 14.814312 38.135014
## 5  2.994894 3.120629 2.742088 2.530772 2.506325 2.867836  4.628244  8.506278
## 6  4.158161 3.643108 3.039841 2.584460 2.498000 2.584443  4.364543  9.164956
##         X17       X18       X19       X20      X21      X22       X23      X24
## 1 24.733121  61.80458  96.91299 102.76175 72.63324 31.92286  7.907782 2.648826
## 2  7.561494  31.08321  62.66402  64.33989 47.62892 22.36920  6.465914 2.718687
## 3 84.352557 125.80971 141.55461 120.86046 69.48430 27.32911  6.250049 2.615079
## 4 75.498082 105.71242 110.08499  84.22875 44.33100 15.91164  4.106571 2.481322
## 5 12.713353  14.05360  11.68812  19.06228 26.87925 24.84912 16.107006 4.704771
## 6 21.380755  37.56913  50.08263  56.10309 51.97472 41.95122 27.279339 9.389539
##        X25       X26       X27       X28       X29       X30       X31
## 1 2.497212  2.997596  7.864413 32.393568  88.90657 151.00972 184.39279
## 2 2.547292  2.531914  2.789033  3.337742  10.03896  49.24866 110.14116
## 3 2.637215  7.647033 33.024593 75.420320 116.90495 137.32745 146.32198
## 4 3.399848 14.180699 47.450849 98.390146 139.89463 158.93809 170.37813
## 5 2.618144  4.871668 14.111769 34.172615  55.73260  49.63522  33.65038
## 6 2.499678  3.388372 11.436350 34.801951  77.54023 115.24432 131.55638
##         X32       X33      X34       X35       X36      X37       X38       X39
## 1 190.75940 166.41014 99.20098 31.224406  3.549203 2.505864  3.960640 22.241325
## 2 113.76635  70.47164 24.37114  5.832823  2.658503 2.466560  2.537436  2.697804
## 3 147.57773 113.01526 55.26608 13.200129  2.639932 2.678911  9.438805 33.116262
## 4 165.32089 113.85595 49.59799  9.323812  2.507768 3.629698 14.518092 35.888571
## 5  53.38664  80.74663 64.24817 34.272313  7.967657 2.790435  7.149218 25.547172
## 6 130.97707 119.35646 99.36712 71.037108 30.828708 2.459464  3.956208 18.547059
##         X40        X41       X42       X43       X44       X45       X46
## 1 79.660749 147.564981 165.51696 142.64359 132.61317 158.45667 147.71885
## 2  3.445632   8.306384  53.47146 145.90033 130.78228  51.89424  10.98181
## 3 56.532570  70.044927  65.86870  75.77462 116.45234 119.74725  66.24304
## 4 54.229818  60.506513  63.09025  94.78636 140.71154 118.30866  54.22400
## 5 68.054793  98.298740  73.64504  38.55474  76.72913 118.03638  76.82529
## 6 60.449313 111.672932 131.34395 115.21153  98.37679  85.59606  71.53883
##         X47       X48      X49      X50       X51        X52       X53
```

```
## 1 69.274468  6.861651 2.444095 8.438945 53.605838 132.140457 160.71120
## 2  3.497954  2.619017 2.486511 2.470630  2.613392   3.240718   7.45371
## 3 16.504402  2.621075 2.587166 6.274568 15.901374  21.409334  22.97109
## 4 10.008209  2.565690 3.187730 6.178080 10.802357  17.739768  37.29297
## 5 26.661596  5.001988 3.057304 9.581940 42.053196 108.481225 125.89195
## 6 54.843973 27.797493 2.517900 5.127063 27.236632  86.695669 143.65034
##         X54       X55       X56       X57       X58        X59       X60
## 1 112.63627  59.60922  46.82929  98.29652 156.74267 102.543360 12.339364
## 2  61.36797 193.34947 117.41743  23.01470   4.00583   2.641628  2.535216
## 3  22.40835  51.29299 113.54009 122.34645  60.24797  12.685257  2.906581
## 4  81.36733 138.87378 158.99894  95.12775  29.97766   5.100224  2.516672
## 5  58.87118  27.88973 108.03580 139.30657  61.36071  12.526473  3.103505
## 6 126.43177  79.52278  52.14138  32.73683  20.68019  15.010177  9.101732
##        X61       X62       X63       X64        X65       X66       X67
## 1 2.451260 18.344154 98.207053 161.72044 124.410186  46.30920  15.43866
## 2 2.477577  2.506718  2.506445   3.30986   6.620486  90.47113 226.75869
## 3 2.513489  3.644525  7.413894  12.48888  20.414114  38.62671  84.21942
## 4 2.605569  3.326099  7.983212  35.58461 104.640736 173.38432 195.40871
## 5 3.219585 14.434839 71.985432 152.17560 126.536849  49.15940  59.83069
## 6 2.554714  5.947712 33.588583 109.33032 172.802990 145.45767 106.95197
##         X68        X69        X70        X71       X72      X73       X74
## 1  18.06167  73.150070 149.190425 115.358518 17.361603 2.485396 36.759427
## 2  74.30727   6.198009   2.629433   2.486297  2.475605 2.516463  2.499428
## 3 130.04040 111.436444  43.131141   8.919869  3.985395 2.555824  6.086369
## 4 163.33507  83.370890  21.989889   3.950346  2.470311 2.713967  4.095285
## 5 159.31126 141.447354  49.171921  11.949180  3.887818 3.384311 22.722657
## 6  77.96758  44.360490  16.233292   5.372305  2.944774 2.538631  5.030868
##          X75        X76       X77       X78        X79       X80        X81
## 1 137.090587 156.724544  71.46132  14.70888   6.594679  20.03695  85.544101
## 2   2.610423   3.249383  11.66842 142.69450 213.876469  32.37066   2.892145
## 3  22.803428  51.749510  84.38541 119.64579 145.973989 140.80858  90.926504
## 4   9.405663  39.098204  96.79281 126.69078 130.447850 139.93991 119.085493
## 5 101.038446 177.955338 161.01790 133.64617 159.506164 205.59005 145.185682
## 6  24.979865  80.964488 122.68643 116.87558 108.877477 102.26624  78.718074
##          X82        X83       X84      X85       X86        X87        X88
## 1 147.969761 105.525902 17.006310 2.504036 54.709273 155.744554 137.663360
## 2   2.541711   2.536444  2.466257 2.476803  2.517502   2.623724   5.149805
## 3  33.140080  12.702039  6.951847 3.804371 22.728171  72.024113 115.990064
## 4  49.188821   9.964802  2.618050 4.535903 10.281711  12.638637  19.008386
## 5  56.991005  19.467097  5.275477 3.039918 17.742647  72.476407 119.993141
## 6  35.353967  10.419379  3.244845 2.655680  8.369482  23.527469  31.827976
##         X89        X90       X91       X92       X93        X94       X95
## 1  43.44588   9.299592  13.96341  54.55621 126.10308 141.189146 76.457147
## 2  39.99240 174.589345 163.31974  19.38725   3.07422   2.565101  2.575957
## 3 144.23251 170.127219 170.98990 142.52092  93.71476  53.944248 32.687569
## 4  27.90712  30.938986  45.41114 100.99465 134.96617  70.819707 17.679938
## 5 131.08939 137.934583 174.35922 179.99561 103.08086  37.339579 11.955777
## 6  41.30685  48.435267  72.02091  96.51037  91.75922  48.652989 16.475341
##         X96      X97       X98        X99      X100      X101      X102
## 1 10.835390 2.604711 56.075835 153.925235 146.60965  68.01979  43.69679
## 2  2.583912 2.503145  2.487502   3.912224  18.02074  79.02624 165.12816
## 3 15.930105 6.649540 45.757692 107.845171 143.28120 162.56214 169.52122
## 4  2.959682 8.164442 26.935027  37.660895  33.74458  23.52870  22.40529
## 5  3.732849 2.848710  6.760061  19.139375  28.03525  36.03793  70.94648
```

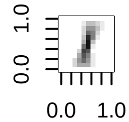
```
## 6  3.768890 2.956927 19.094509  54.253775  47.84539  35.76858  43.85592
##        X103      X104       X105       X106      X107      X108     X109
## 1  71.01945 125.93443 151.010198 106.437540 38.285523  4.953662 2.495487
## 2 118.94927  22.62569   4.517744   3.055216  2.685046  2.639426 2.531738
## 3 157.28053 136.62854 116.974321  91.407198 59.911780 21.005701 6.976369
## 4  48.66412 114.62938 134.440380  69.835338 17.767540  2.791276 9.190923
## 5 123.78115 117.96700  56.619691  17.485111  5.339111  2.708417 2.531318
## 6  76.31224 105.16088  93.154126  48.987793 17.647402  3.748399 2.818490
##        X110       X111       X112      X113      X114      X115      X116
## 1 35.242206 124.402944 181.376468 167.14195 150.69574 162.81310 160.73674
## 2  3.199127  10.594941  40.697398  99.40311 134.51240  99.54037  29.75983
## 3 45.242628 106.593070 157.073639 166.87529 140.46992 110.37293 101.18694
## 4 39.495430  78.689363  99.355402  93.49060  97.56674 132.00416 155.35270
## 5  3.064400   4.370402   8.754828  27.89119  69.50126  96.82846  88.00162
## 6 17.140014  74.474492 109.003606 107.37177 106.98442 121.01789 123.60860
##         X117      X118      X119      X120     X121      X122      X123
## 1 111.815480 48.456863 10.892800  2.744869 2.500525  9.443734 50.162696
## 2   6.541923  3.399184  2.645063  2.594525 2.500235  4.162405 15.387226
## 3 100.403317 82.502117 46.726615 14.213326 3.530669 15.450675 43.582441
## 4 112.804262 43.943946  8.274893  2.635275 5.184964 22.514451 66.770961
## 5  45.850972 15.216082  4.891293  2.746305 2.522461  2.509027  4.129071
## 6  83.988731 37.442126 11.682172  2.911542 2.585135  7.311689 37.344736
##        X124      X125      X126      X127      X128     X129      X130
## 1 117.77538 166.09238 169.50622 136.29046  81.71880 33.57678  8.835667
## 2  47.95953  83.45215  95.21241  79.70692  28.31256  5.59683  3.070167
## 3  68.69839  70.44751  54.00391  41.39247  38.64506 36.74820 28.487732
## 4 125.40502 165.02203 174.26116 154.70763 103.92459 45.36520 11.975590
## 5  12.79447  36.48984  64.51875  75.77959  68.87727 40.49312 16.151380
## 6  90.31732 130.94258 143.40078 130.86604  91.79188 43.06794 14.980065
##        X131     X132     X133     X134      X135      X136      X137      X138
## 1  3.208758 2.544033 2.509593 2.588867  4.247129  9.781316 17.537750 19.721763
## 2  2.506383 2.539672 2.474766 2.693306  4.085078  7.921546 11.628430 12.807547
## 3 15.037421 4.984900 2.558420 2.600222  3.196815  4.078344  4.460521  4.981771
## 4  3.222472 2.518299 2.840472 5.674803 13.456497 27.706007 40.533924 41.448563
## 5  4.949614 2.708112 2.505536 2.496977  3.581878  7.696682 16.822656 25.150354
## 6  4.432850 2.541367 2.465798 2.765725  6.952213 19.422205 33.740763 39.764694
##        X139      X140      X141     X142     X143     X144
## 1 14.452282  7.212133  3.453959 2.658262 2.585022 2.496620
## 2 11.896035  4.984525  2.635044 2.514537 2.509231 2.518327
## 3  4.902579  4.361451  4.034837 3.520218 3.091879 2.658653
## 4 30.531943 15.547873  5.795940 2.794524 2.482373 2.467381
## 5 29.415302 24.400766 14.431607 6.675635 3.311799 2.540618
## 6 30.596167 15.961913  7.093386 3.517541 2.696308 2.557159
```

```r
par(mfrow = c(3,2))
for (i in 1:10){
  plot_digit(meanPixel[i,], bw = FALSE, main = paste("digit:", i - 1))
}
```

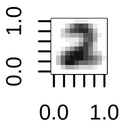
digit: 0



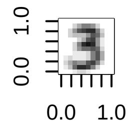
digit: 1



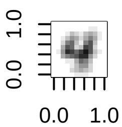
digit: 2



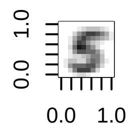
digit: 3



digit: 4

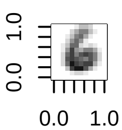


digit: 5

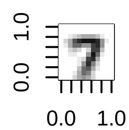



```
par(mfrow = c(1,2))
```

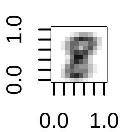
digit: 6



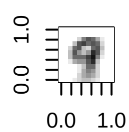
digit: 7



digit: 8



digit: 9



Judging the graph, **0**, **5**, **2** seems to have less within class variation over the image.It is likely that our model will perform well in predicting these digits.

On the other hand, digits like **1**, **4**, **6**,**8**, **9** display a relatively large within class variation over the image, meaning that it can be hard for the model to classfy these digits.

Part 3 (5 pts.)

Using the average images, come up with a measure of similarity or dissimilarity between different digits. It doesn't need to be particularly elegant.

Which pairs of digits will be easiest to tell apart? Which pairs will be most difficult? Thinking about how people write different digits, does your similarity measure make sense?

Rules One measure of that we can leverage to classify different digit can be measure coordinate that appear to be the darkest for a given graph. The reasoning behind is that different digits has different emphasis (where we dedicate most the ink to) on strokes.

Hence, if a large amount of samples all shares similar darkest coordinate, it is highly likely that these samples belong to the same group.

Question 2 (15 pts.)

Let's start with one of the digit pairs that is easiest to tell apart: 0s and 1s. For this question, we'll build a classifier to discriminate between images of 0s and images of 1s. Start by subsetting your training and validation data sets to only include 0s and 1s (a literal manifestation of the classification problem).

Part 1 (8 pts.)

For reasons you will soon see, we'll only consider one classification approaches for this problem - logistic regression. Using your training data, train a logistic regression classifier for the literal 0/1 problem. Compute a 10-fold CV measure of expected prediction accuracy using the training data. Similarly, compute the accuracy of your trained logistic regression model on the validation set. What do you find here?

Intuitively, explain this result. Think carefully about the how logistic regression is approaching this classification problem.

Plot up to 4 images in the validation set that are misclassified with respect to the Bayes' classifier. Does this misclassification make sense? What about these images leads the classifier to incorrectly guess the proper label?

Note: the underlying algorithm for logistic regression proceeds iteratively attempting to minimize the logistic regression loss function. Because many of the predictors in this problem have variance close to zero, the default number of iterations (typically 25) may not be enough for the algorithm to converge. You can deal with this issue by setting the maximum number of IRLS iterations to a larger value - 100 should be sufficient. In R, this can be achieved within your `glm` call by adding an additional control argument - `control = list(maxit = 100)`.

Part 2 (7 pts.)

Let's try to understand exactly how this classifier is working. For the training data, compute a logistic regression classifier with the LASSO penalty on the coefficients. Use K-fold CV to find a value of λ that sparsely represents the set of coefficient and viably minimizes the expected misclassification rate for out of sample data.

Using the coefficients associated with your chosen value of λ, create a plot that shows the relationship between the pixels and the coefficients. Positive coefficients are associated with pixels where a pencil pixel in that location (value > 0) **increases** the predicted probability that the image is a 1 while negative coefficients are associated with pixels where a pencil pixel in that location (value > 0) **decreases** the predicted probability that the image is a 1.

Leveraging this plot, come up with a set of rules related to the location of pencil pixels that a human who had the pixel mappings could evaluate to determine if an image is a zero or one. The rules don't need to exactly relate to specific pixels. Rather, they can relate to relative location of the pixels (center vs. noncenter, for example).

This task is, more or less, an example of **computer vision** - trying to use classification methods to make computer view images in the same way as humans.

Question 3 (25 pts.)

Now, let's work with a more difficult pair - 4s and 9s. Logically, these are going to be more difficult to distinguish! Start by subsetting your training and validation sets to only include 4s and 9s. There may be some situations where you need to recode these to zeros and ones! When making predictions on the test set, be sure to recode the predictions back to 4s and 9s.

Part 1 (5 pts.)

Start by replicating your analysis for 0s and 1s using logistic regression for 4s and 9s. Compute a 10-fold CV measure of expected prediction accuracy and find the misclassification rate for your validation set. How does this compare to the performance of logistic regression for 0s and 1s? Why do these results differ?

Using the same LASSO approach as above, create a plot that shows which pixels are important for the classification. How does this image differ from the 0/1 case?

Part 2 (5 pts.)

Compute QDA and Naive Bayes classifiers for the 4/9 problem. Compare their performance to your logistic regression. What's going on here? Why do we see what we see? Think carefully about the assumptions under-the-hood for QDA, Naive Bayes, and logistic regression and the structure of the pixel data in the MNIST data set.

Part 3 (10 pts.)

Using the full suite of classification approaches discussed in class, find a classification approach that **minimizes** the expected misclassification rate of 4s and 9s on a true out of sample data set.

Your answer should discuss the possible approaches to this problem and explain how you made your final choice. Discuss how you chose any tuning parameter values.

You do not need to run all possible classification methods to get full credit for this question! There are some methods that we can rule out without ever running them. When doing this, provide grounded reasoning related to the strengths and weaknesses of different approaches.

For your chosen method, explain **why** it outperforms all other approaches. Think carefully about strengths and weaknesses.

Finally, present at least 4 examples of misclassified images. Could we ever expect a classification algorithm to get those images correct?

Part 4 (5 pts.)

Use the hidden test set to generate a prediction for each included image. Your predictions should be stored in a matrix that has the image key as the first column and the integer value of the class in the second column. Save this matrix as `Q3Predictions.csv` and include it with your final submission.

Points for this question will be given with respect to classification accuracy. Let E_i be the proportion of observations in the test set (that are actually 4s and 9s) misclassified. Let E_{min} be the minimum proportion of misclassified observations across the class. Then, you final point total for this part will be:

$$5 \times \frac{E_{max}}{E_i}$$

Question 4 (20 pts.)

Now, let's work with **three classes** - 3s, 5s, and 8s. Start by subsetting your training and validation sets to only include 3s, 5s and 8s.

Part 1 (5 pts.)

Let's start with multinomial logistic regression. Compute a 10-fold CV measure of the expected misclassification error and compute the error rate on the validation set. How does this compare to your previous two-class analyses?

Using the **posterior probabilities** that each observation in the validation set belongs to each class, find 2 images that are close to each decision boundary: 3/5, 3/8, 5/8, and 3/5/8. Plot these images and discuss any factors discriminating between these three digits that multinomial logistic regression is missing. Hint: Think about the contextual aspects that tell a human that a digit is a digit.

Note: As with logistic regression, multinomial logistic regression iteratively optimizes the multinomial logistic loss function. The default of 100 iterations is likely not enough. You can change this in `nnet::multinom` by adding an argument `maxit = 1000`.

Part 2 (10 pts.)

Using any information gained from Part 1 to your advantage, find a classification approach that **minimizes** the expected misclassification rate of 4s and 9s on a true out of sample data set.

Your answer should discuss the possible approaches to this problem and explain how you made your final choice. Discuss how you chose any tuning parameter values.

You do not need to run all possible classification methods to get full credit for this question! There are some methods that we can rule out without ever running them. When doing this, provide grounded reasoning related to the strengths and weaknesses of different approaches.

For your chosen method, explain **why** it outperforms all other approaches. Think carefully about strengths and weaknesses.

Finally, present at least 4 examples of misclassified images. Could we ever expect a classification algorithm to get those images correct?

Part 3 (5 pts.)

Use the hidden test set to generate a prediction for each included image. Your predictions should be stored in a matrix that has the image key as the first column and the integer value of the class in the second column. Save this matrix as `Q4Predictions.csv` and include it with your final submission.

Points for this question will be given with respect to classification accuracy. Let E_i be the proportion of observations in the test set (that are actually 3s, 5s, and 8s) misclassified. Let E_{min} be the minimum proportion of misclassified observations across the class. Then, you final point total for this part will be:

$$5 \times \frac{E_{max}}{E_i}$$

Question 5 (25 pts.)

Finally, let's work with the full MNIST data set. This is a 10 class classification problem that has been studied extensively. With your work on this problem, you will join the club of data scientists who have taken a crack at one of machine learning's most infamous classification tasks!

Part 1 (15 pts.)

Use any tools in our classification arsenal to try to minimize the expected misclassification rate for out of sample handwritten digits.

In your answer, you should benchmark any algorithms that are suited for the problem. For any classification methods that you know will not work well (by virtue of the structure of the data), justify why it's not worth the time to check it. Be sure to explain your method for tuning any hyperparameters.

For any models you run, produce a table that shows the misclassification rate on the validation set. Similarly, record the compute time needed to arrive at your final model (including any hyperparameter tuning) and include this in the same table. See this StackOverflow thread for an elementary way to do this in R.

Which model performs the best on the 10 class problem? Why do you think this model performs the best? Compare your approach to other possible approaches when answering this question.

Is the tradeoff in accuracy vs. compute time worth the gain in realistic scenarios? Think about how these algorithms might scale as N and P get larger. Specifically, discuss the problem of hyperparameter tuning and how this might contribute to difficulty in implementing your chosen approach.

For one-off classification problems, the computational time may not matter. However, there is significant research into the area of **online classification algorithms** - algorithms in which new predictions can be made quickly using existing data and the classification algorithm can be updated using new training data as it arrives. See this Wikipedia page for more information on this topic.

Part 2 (5 pts.)

For your final model, compute a **confusion matrix** for the validation set that shows how often each digit is classified in each class. Which incorrect classifications happen most frequently?

For the most commonly confused digit pairs, plot examples that are misclassified. What aspect of these images led to their misclassification?

The MNIST data as presented to you has been simplified in some ways. There are also other data cleaning tasks that can improve predictive accuracy. What are some steps that could be taken in the data cleaning stage that would help your chosen method improve its misclassification rate?

Part 3 (5 pts.)

Use the hidden test set to generate a prediction for each included image. Your predictions should be stored in a matrix that has the image key as the first column and the integer value of the class in the second column. Save this matrix as `Q5Predictions.csv` and include it with your final submission.

Points for this question will be given with respect to classification accuracy. Let E_i be the proportion of observations in the test set misclassified. Let E_{min} be the minimum proportion of misclassified observations across the class. Then, you final point total for this part will be:

$$5 \times \frac{E_{max}}{E_i}$$