Midterm Data Exercise #2 - Classification

Kevin McAlister

March 28th, 2022

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!

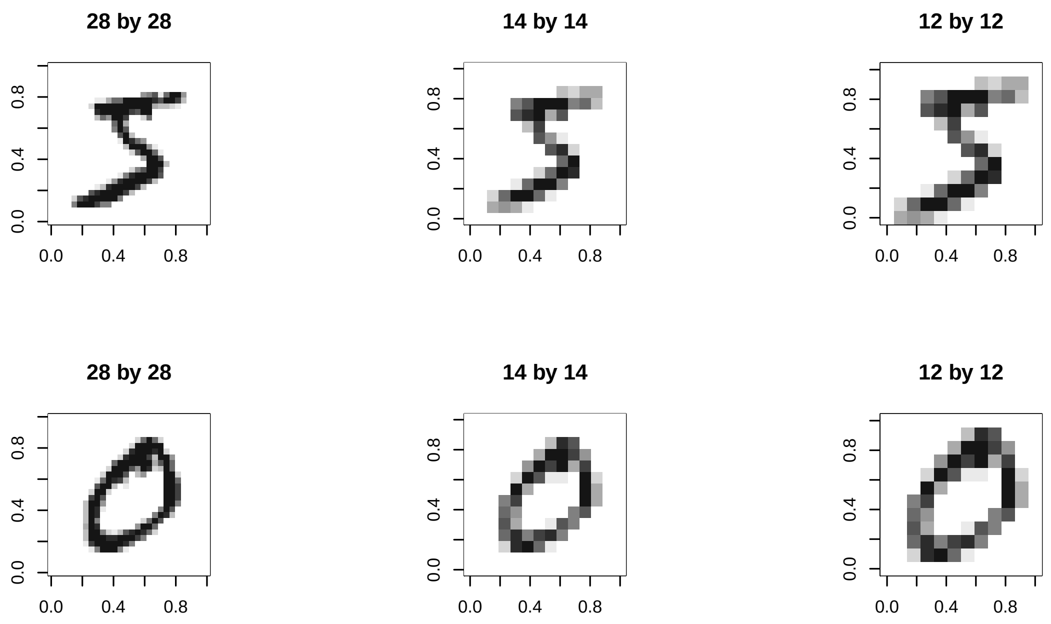
Modifying the MNIST Data Set

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:

```r
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),
        ...)
}
# Example
plot_digit(x = train_x[1, ], bw = FALSE, main = "True Class = 0")
```

`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?

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?

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?

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:

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