Kevin Shao & Jeffrey Cheng
Mr. Piper
RML B & F
23 May 2018

Proposal for Dimension Reduction

1 Methodology

We had two ideas for dimension reduction of our variables. One was averaging the pixel intensities of the 28x28 image to create a 7x7 with less pixels. Another was simply taking out the pixels that were completely correlated with one another (all intensities were zero for that one pixel for each image in training set). These two methods illustrated above are the "visual" was of thinking about dimensional reduction. Both are a type of PCA, where we essentially take linear combinations of the pixel intensities. However, we also plan to run PCA using R and determine the "mathematical" best way to reduce our dimensions.

2 Pixel Intensity Averaging

Pixel subsampling, in practice, becomes challenging when considering the structure of our data. Namely, our train and test data comes in large matrices, where the i-th observation (i-th row) consists of 785 observations, where the latter 784 are the pixels, we have little information regarding the position of pixels.

Our goal was to subsample from a 28x28 pixel image to a 14x14 pixel image by averaging. That is, for each 2x2 pixel group, we would take the four pixel intensities and take their mean. Intuitively, this would result in a similar, albeit lower quality, image.

```
pixel_positions = matrix(nrow=196,ncol=4)
for (i in 1:14) {
  for (j in 1:14) {
    pixel_positions[i+14*j-14,1] = (i-1)*2 + (j-1)*56 + 1
    pixel_positions[i+14*j-14,2] = (i-1)*2 + (j-1)*56 + 2
    pixel_positions[i+14*j-14,3] = (i-1)*2 + (j-1)*56 + 29
    pixel_positions[i+14*j-14,4] = (i-1)*2 + (j-1)*56 + 30
  }
}
```

The approach we took was first to create a new matrix, called "pixel positions", consisting of 196 rows (since there will be 196 pixels in the new image) and 4 columns (since each pixel in the new image corresponds to 4 pixels in the old image. Essentially, the i-th row consisted of the pixels from the old image to average to obtain the i-th pixel in the new image.

```
test_sub = matrix(nrow = 9999, ncol = 196)

for (i in 1:196) {
  test_sub[,i] = test[,c(pixel_positions[i]) + 1 ]
}

train_sub = matrix(nrow = 59999, ncol = 196)

for (i in 1:196) {
  train_sub[,i] = train[,c(pixel_positions[i]) + 1]
}
```

We then created matrices to hold the subsampled train and test data sets, called "train_sub" and "test_sub", respectively. For each of the 196 **columns** (hence "test_sub[,i]"), we wanted the average of the columns (of test or train) that were given by the i-th entry in our list. The increment at the end of the index serves to eliminate the leading column of the original data, which contains the true value of the image. We now arrive at two matrices of desired content.

Having attempted reclassification with KNN, we find that a mere 2x2 subsampling is insufficient for meaningful time change. Thus, we modified the program to take 4x4 subsampling as follows, with the same intent:

```
pixel_positions = matrix(nrow=49, ncol=16)
for (i in 1:7) {
  for (j in 1:7) {
    pixel_positions[i+7*j-7,1] = (i-1)*4 + (j-1)*112 + 1
    pixel_positions[i+7*j-7,2] = (i-1)*4 + (j-1)*112 + 2
    pixel_positions[i+7*j-7,3] = (i-1)*4 + (j-1)*112 + 3
    pixel_positions[i+7*j-7,4] = (i-1)*4 + (j-1)*112 + 4
    pixel_positions[i+7*j-7,5] = (i-1)*4 + (j-1)*112 + 29
    pixel_positions[i+7*j-7,6] = (i-1)*4 + (j-1)*112 + 30
    pixel_positions[i+7*j-7,7] = (i-1)*4 + (j-1)*112 + 31
    pixel_positions[i+7*j-7,8] = (i-1)*4 + (j-1)*112 + 32
    pixel_positions[i+7*j-7,9] = (i-1)*4 + (j-1)*112 + 57
    pixel_positions[i+7*j-7,10] = (i-1)*4 + (j-1)*112 + 58
    pixel_positions[i+7*j-7,11] = (i-1)*4 + (j-1)*112 + 59
    pixel_positions[i+7*j-7,12] = (i-1)*4 + (j-1)*112 + 60
    pixel_positions[i+7*j-7,13] = (i-1)*4 + (j-1)*112 + 85
    pixel_positions[i+7*j-7,14] = (i-1)*4 + (j-1)*112 + 86
    pixel_positions[i+7*j-7,15] = (i-1)*4 + (j-1)*112 + 87
    pixel_positions[i+7*j-7,16] = (i-1)*4 + (j-1)*112 + 88
  }
}
for (i in 1:49) {
  test_sub[,i+1] = test[,c(pixel_positions[i]) + 1 ]
}

train_sub = matrix(nrow = 59999, ncol = 50)

for (i in 1:49) {
  train_sub[,i+1] = train[,c(pixel_positions[i]) + 1]
}
```

Then, re-running the KNN classification code, we find a massive speed boost. Whereas the original would take 5-6 seconds to classify each case, the reduced images can be classified at an average rate of 3 per second, a 15-fold increase in speed.

Unfortunately, by subsampling so much, we inevitably lose data, and with it, accuracy. Classifying it with the same KNN algorithm, with the same k parameter of 7, we get a test error of 18.60%, 6 times as much as the original.

3 Removing Collinear Pixels

Removing collinear pixels was pretty simple because we had accomplished this once already when doing LDA and QDA. Essentially, we are taking a linear combination of the pixels, with each weight being a zero or a one based on whether we decide to remove it or not. In doing this dimension reduction, we are not actually running PCA on the dataset, but rather taking out columns which do not contribute to the overall image as much according to us. Therefore, if the average pixel intensity of a specific pixel over the 59999 training images is less than 10, we removed

the pixel from the reduced training set.

We reduced our number of variables down from 784 to 370, which is a significant decrease. The runtime for a test set of 200 images accordingly decreased from 49 seconds to 23 seconds. Interesting enough, we expected an increase in our test error since we included less explanatory variables. However, our test error actually decreased, from 0.04975124 to 0.04477612.

4 Mathematical Dimensional Reduction

By taking advantage of the R's library MASS, we can reduce the dimensionality of our dataset based on their importance. However, an important caveat is that we must first combine the training and test sets to do PCA. After all, nothing would work if we determined different principal components for each of the different sets.

Then we write:

```
ktrain = subset(rawtrain,select= -c(1))
ktest = subset(rawtest,select= -c(1))
full = rbind(ktrain, ktest)
pca = prcomp(full)
pcatrain = data.frame(trainlabel, pca$x[1:59999,c(1:20)])
pcatest = data.frame(testlabel, pca$x[60000:69998,c(1:20)])
```

The first two lines remove the first column of each set (the labels). The third line concatenates them together, and the fourth performs PCA. The fifth and sixth lines, finally, re-divide them and re-prepend the labels. This process is rather long, however; the PCA step took around 5 minutes.

However, the results are astounding. When we re-run KNN on this new set of pcatrain and pcatest, keeping the k parameter of 7, we find a test error rate of 0. That is, none of the test set was misclassified.

5 Conclusion

Between the three dimension reduction methods we investigated, mathematical dimension reduction, PCA, was clearly the best in terms of accuracy. It did take a relatively long time to run. However, when compared to the runtime of classification of the original and its accuracy, it is far superior. Removing collinear pixels, while sensible and beneficial, provided relatively little benefit, only halving the number of predictors and improving error marginally. Pixel intensity averaging, while rather intuitive, loses information (as you would expect when decreasing the quality), and thus made our test error far worse despite speeding up the classification step greatly.