LDA & QDA on MNIST data with PCA

Chris Schmidt

10/19/2019

Evaluating Linear Disciminant Analysis and Quantitative Discriminant Analysis on the MNIST Data using PCA to Determing the Optimal Attributes.

We build and evaluate LDA and QDA to build predictive models to select the correct number using the MNIST data set. This project uses Principal Component Analysis to select the most significant predictors to build the models.

Install library(MASS) We will need several tools in the MASS package so we need to load and install it.

```
library(MASS)
```

Import data set "train.csv" The MNIST data set is comprised of a large number of black and white images of handwritten digits. The dataset we are using comes from the .csv file named train.csv that we read into our program using the *read.csv()* function as shown below. There are 42,000 observations of 785 variables in our data set. We output the structure, the head and tail of the data to illustrate the composition.

```
mnist <- read.csv('train.csv', header=T)

# the following descriptive functions are commented out as they take up
# a large number of pages in the output when knitting to .pdf.

#str(mnist)
#head(mnist)
#tail(mnist)
```

Standardize the dataset we want to standardize the data which we accomplish by dividing the difference between attribute values and the minimum attribute value by the difference between the maximum and minimum attribute values as shown in the code block below.

```
mnist[,2:784] <- (mnist[,2:784] - min(mnist[,2:784])) / (max(mnist[,2:784]) - min(mnist[,2:784]))
```

Create the training and test datasets We build two functions, the first with the pixel column values greater than zero and the second creating a very large (and very sparse) matrix of values. We then split the matrix we created into test and training sets and transform the test_data into a data frame.

```
BB <- mnist[, 2:784]>0
one <- apply(BB, 2, as.integer)

train_mnist <-one[1:10000,1:278]
test_mnist <- one[10001:42000,1:278]
test_data <- data.frame(test_mnist)
```

#Use Principal Component Analysis using the prcomp() function to reduce the dimensionality of the dataset. Principal Component Analysis (PCA) is a dimensionality reduction technique used to derive a low-dimensional set of features from a larger, sometimes much larger set of variables.

PCA is used to reduce the dimensions of a $n \times p$ data matrix X. The *first principal component* direction of the data is that along which the observations have the largest variance. This can be interpreted as defining the line that is as close as possible to the data. The first principal component line minimizes the sum of the squared orthogonal distances between each point and the line. The second principal component is a linear combination of variables that are uncorrelated with the first principal component and has the largest variance subject to this constraint.

The general idea behind dimension reduction methods is that the predictors are *transformed* and then fit to a least squares model.

All dimension reduction techniques work in two steps: First the transformed predictors $Z_1, Z_2, ..., Z_M$ are derived. Second, the model is fit using these M predictors. There are several ways to do this but we are interested in PCA as described above.

If we let $Z_1, Z_2, ..., Z_m$ represent $M < p$ *linear combinations* of our original p predictors. So that we have

$$Z_m = \sum_{j=1}^{p} \phi_{jm} X_j$$

for some constants $\phi_{1m}, \phi_{2m}, ..., \phi_{pm}$, for $m = 1, ..., M$. We can then fit the linear regression model

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \epsilon_i, \text{ for } i = 1, ..., n$$

using the least squares method. If the constants $\phi_{1m}, \phi_{2m}, ..., \phi_{pm}$ are properly chosen given the regression coefficients $\theta_0, \theta_1, ..., \theta_M$ dimensionality reduction approaches can perform least squares regression.

We reduce the problem of estimating the $p + 1$ coefficients $\beta_0, \beta_1, ..., \beta_p$ t the simpler problem of estimating the $M + 1$ coefficients $\theta_0, \theta_1, .., \theta_M$ where $M < p$ thus reducing the problem from $p + 1$ to $M + 1$.

Note that

$$\sum_{m=1}^{M} \theta_m z_{im} = \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{jm} x_{ij} = \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{jm} x_{ij} = \sum_{j=1}^{p} \beta_j x_{ij}$$

where

$$\beta_j = \sum_{m=1}^{M} \theta_m \phi_{jm}$$

Applying PCA to the MNIST data set. We are interested in reducing the number of attributes we need to deal with by using PCA to determine which are the most significant contributors to our labels.

Using the *prcomp()* function on the mnist data set and excluding the label column, we derive our ordered principal components in our pca_mnist

```
pca_mnist= prcomp(mnist[,-1])
dim(pca_mnist$x)
```

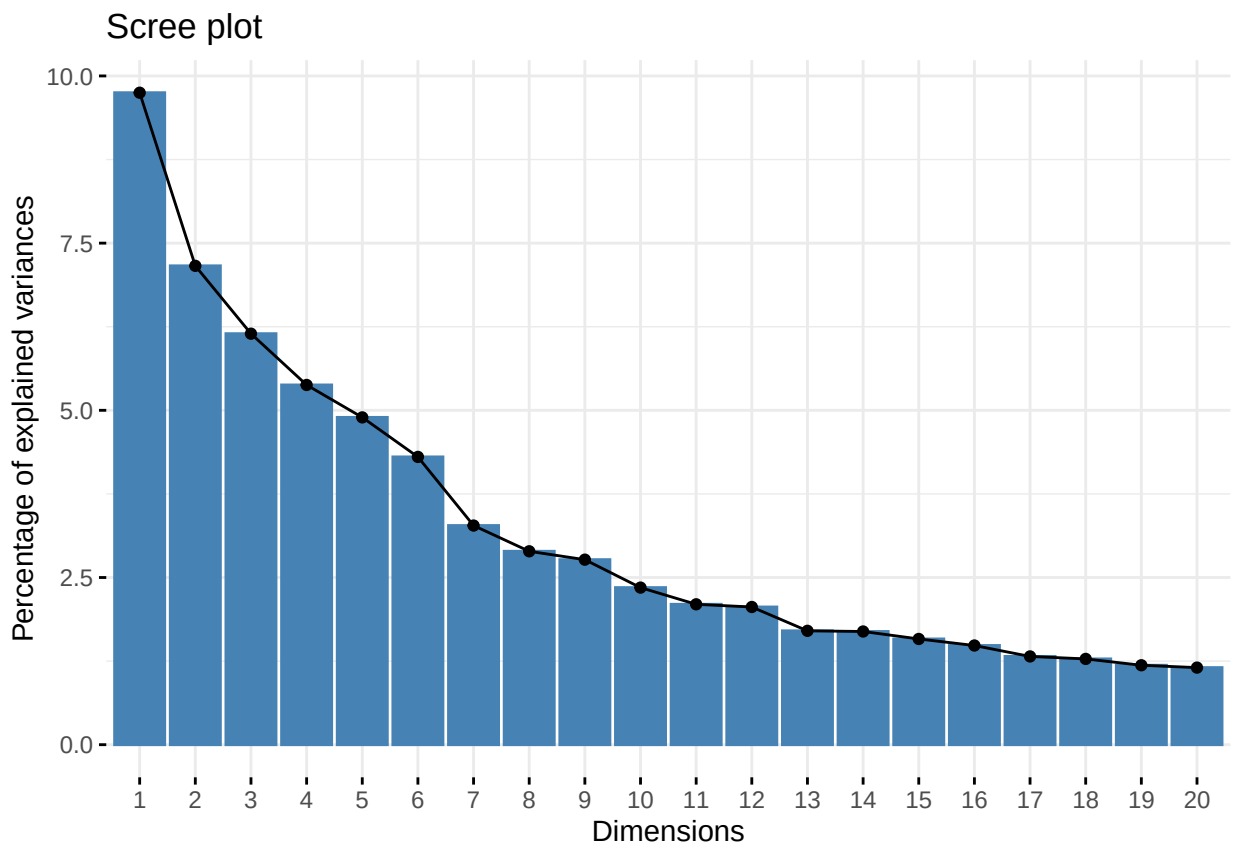
```
## [1] 42000   784
```

```
library(factoextra) # load to compute scree and eigenvector plots
```

```
## Loading required package: ggplot2
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
fviz_eig(pca_mnist, ncp=20)
```



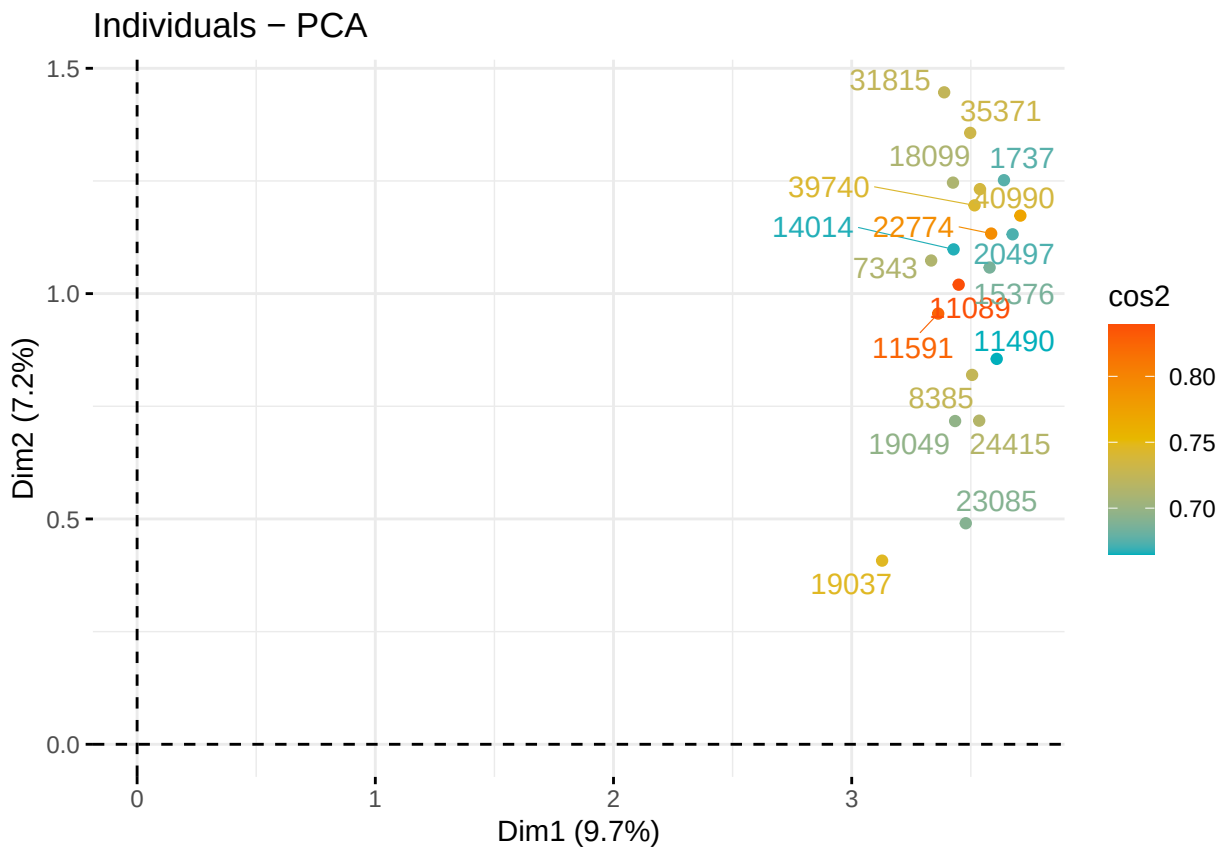

```
train_mnist <- data.frame(pca_mnist$x[1:10000,1:60])
test_mnist <- data.frame(pca_mnist$x[10001:42000,1:60])
test_data <- data.frame(test_mnist)
```

We can see from the scree plot that the first 10 principal components contribute a significant amount of the information in the data. This plot shows the percentage of variances explained by each of the first 20 principal components.

We can also look at a plot of the top 20 principal components on a grid showing the percentage contribution of the 1st and 2nd component and the location details in relationship to those. In addition we have a heat map style that indicates the significance of a particular component to the data.

```
fviz_pca_ind(pca_mnist, col.ind = "cos2",
   repel = TRUE, gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
   select.ind = list(cos2 = 20))
```

```
## Warning: ggrepel: 1 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```



Build the LDA model to recognize '0'. Linear discriminant analysis models the probability distribution of the predictors X separately in each of the response classes Y, i.e. we want to find $Pr(X = x|Y = k)$ and Bayes Theorem is used to flip these around into estimates for $Pr(Y = k|X = x)$. When the distributions are Gaussian this model is very close in form to logistic regression. The reasons for using LDA over logistic regression include the facts:

- when the classes are well separated, the parameter estimates for the logistic regression model can be unstable.
- if n is small and the distribution of the predictors $X \sim N(\mu, \sigma^2)$, LDA is more stable than logistic regression.
- If we have more than two response classes Y, LDA is more popular.

Using Bayes Theorem for Classification If $k \geq 2$ and we want to classify an observation into one of K classes where the qualitative response variable Y can take on one of K distinct and unordered values. We let π_k be the *prior* probability that a randomly chosen observation comes from the kth class and let $f_k(x) \equiv Pr(X = x|Y = k)$ be the density function of X for an observation from the kth class. $f_k(x)$ is relatively large if there is a high probability that an observation in the kth class has $X \approx x$ and $f_k(x)$ is relatively small if it is very unlikely that an observation in the kth class has $X \approx x$.

Bayes Theorem states that

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

Letting $p_k(x) = Pr(Y = k|X)$ we see that we can simply plug in estimates of π_k and $f_k(X)$ into the formula which can be generated with the software that then takes care of the rest. We refer to $p_k(x)$ as the *posterior*

probability that an observation $X = x$ belongs to the kth class given the predictor value for that observation.

Estimating π_k is easy if we have a random sample of Y's from the population but estimating $f_k(X)$ is more difficult. However, if we have an estimate for $f_k(x)$ then we can build a classifier that approximates the Bayes classifier.

By assuming that $X = (X_1, X_2, ..., X_p)$ is drawn from a multivariate Gaussian distribution, with a class specific mean vector and a common covariance matrix which we can write as $X \sim N(\mu, \Sigma)$ to indicate that p has a multivariate Gaussian distribution. $E(X) = \mu$ is the mean of the X vector with p components and $Cov(X) = \Sigma$ is the $p \times p$ covariance matrix of X.

Formally, the multivariate Gaussian density is

$$f(x) = \frac{1}{(2\pi)^{p/2}|\Sigma|^{1/2}} exp\Big(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\Big)$$

plugging the density function for the kth class, $f_k(X = x)$ into

$$Pr(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{K} \pi_l f_l(x)}$$

and applying some algebra we see that the Bayes classifier assigns $X = x$ to the class for which

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k$$

is the largest. The Bayes decision boundaries represent the set of values x for which $\delta_k(x) = \delta_l(x)$. In other words for which

$$x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k = x^T\Sigma^{-1}\mu_l - \frac{1}{2}\mu_l^T\Sigma^{-1}\mu_l, \text{ for } k \neq l$$

The $\log \pi_k$ term has disappeared because each of the three classes has the same number of training observations, thus π_k is the same for each class. To estimate $\mu_1, ..., \mu_k, \pi_1, ..., \pi_k$ and Σ we use similar conventions for the case where $p = 1$

$$\hat{\mu}_k = \frac{1}{n_k}\sum_{i:y_i=k} x_i$$

$$\hat{\Sigma} = \frac{1}{n-K}\sum_{k=1}^{K}\sum_{i:y_i=k}(x_i - \hat{\mu}_k)^2$$

$$\hat{\pi}_k = \frac{n_k}{n}$$

The estimates are plugged into

$$\delta_k(x) = x^T\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma^{-1}\mu_k + \log \pi_k$$

in order to assign a new observation $X = x$ to the class for which $\hat{\delta}_k(x)$ is the largest. This is a linear function of x so the LDA decision rule depends on x only through a linear combination of the elements.

The output for LDA often uses a *confusion matrix* to display the True status versus the predicted status for the qualitative response variable. Elements on the diagonal represent correct predictions and off-diagonal represent misclassifications.

This introduces the terms *sensitivity* and *specificity* to characterize the performance of a classifier. Sensitivity is the percentage of correctly specified positive responses identified while specificity is the percentage of correctly specified negative responses that are identified. We call the true positive rate the *sensitivity* and the false positive rate, 1 - *specificity*

The Bayes classifier works by assigning an observation to the class for which the posterior probability $p_k(X)$ is the largest. If we have two classes, say "*wrong*" and "*right*" we assign the observation to the "*wrong*" class if

$$Pr(\text{ wrong} = \text{Yes } |X = x) > 0.5$$

This creates a threshold of 50% for the *posterior* probability of default in order to assign an observation to the "*wrong*" class. If we have concerns about mislabeling the prediction for the "*wrong*" class we can lower this threshold. We could, for example, label an observation with a posterior probability of being in the "*wrong*" class about 20% to the "*wrong*" class

$$Pr(\text{ wrong} = \text{Yes } |X = x) > 0.2$$

We use the *receiver operating characteristics* curve, ROC curve, to simultaneously display the two types of errors for all possible thresholds where the overall performance of the classifier is given by the area under the ROC curve (the AUC) where the larger the percentage, the better the classifier.

Build the LDA model to evaluate the PCA Test and Train Data to Identify "0"

We build the LDA model using the *lda()* function with 'label' as the response and using the training data set 'train_data' as the predictor set. By setting *y_t* to be match the first column to be equal to 0 and then building the training and test sets we can build the LDA to recognize "0" and output the accuracy using *mean()* function applied to the predictions generated by the *predict()* function on the model using the test data.

```
y_t <- (mnist[,1] == 0)
y_train <- y_t[1:10000]
y_test <- y_t[10001:42000]

train_data = data.frame(label = y_train, train_mnist)


# Fit the model
m_lda <- lda(label~.,data = train_data)

lda.pred <- predict(m_lda , test_mnist)

mean(lda.pred$class == y_test)
```

```
## [1] 0.9838438
```

Figure out what to do with the rest of the digits and then build out the QDA model QDA assumes the observations come from a Gaussian distribution like LDA but QDA assumes each class has its own covariance matrix. QDA assumes that an observation from the kth class is of the form $X \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the kth class.

In this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\Sigma_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \log|\Sigma_k| + \log\pi_k\end{aligned}$$

is the largest. The QDA classifier plugs estimates for Σ_k, μ_k, and π_k into the equation above and then assigning $X = x$ to the class for which the quantity is largest. Since x appears as a quadratic equation, we call this classifier QDA.

The reasons for choosing LDA over QDA or vice versa have to do with the bias-variance tradeoff. When there are p predictors, estimating the covariance matrix requires estimating $p(p+1)/2$ parameters. QDA estimates a separate covariance matrix for each class for a total of $Kp(p+1)/2$ parameters. If we have 50 predictors this is some multiple of $(50 * 51)/2 = 1275$ for a significant jump in predictors. The LDA model assumes the K classes share a common covariance matrix so that the LDA model becomes linear in x so that there are kp linear coefficients to estimate. Thus LDA is a less flexible classifer than QDA and has a significantly lower variance. The tradeoff comes from noting that if the LDA assumption of a common covariance matrix is incorrect then high bias can be an issue.

Build QDA models to evaluate the PCA Test and Train Data

QDA assumes the observations come from a Gaussian distribution like LDA but QDA assumes each class has its own covariance matrix. QDA assumes that an observation from the kth class is of the form $X \sim N(\mu_k, \Sigma_k)$, where Σ_k is a covariance matrix for the kth class.

In this assumption, the Bayes classifier assigns an observation $X = x$ to the class for which

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}\log|\Sigma_k| + \log\pi_k \\ &= -\frac{1}{2}x^T\Sigma_k^{-1}x + x^T\Sigma_k^{-1}\mu_k - \frac{1}{2}\mu_k^T\Sigma_k^{-1}\mu_k - \log|\Sigma_k| + \log\pi_k\end{aligned}$$

is the largest. The QDA classifier plugs estimates for Σ_k, μ_k, and π_k into the equation above and then assigning $X = x$ to the class for which the quantity is largest. Since x appears as a quadratic equation, we call this classifier QDA.

The reasons for choosing LDA over QDA or vice versa have to do with the bias-variance tradeoff. When there are p predictors, estimating the covariance matrix requires estimating $p(p+1)/2$ parameters. QDA estimates a separate covariance matrix for each class for a total of $Kp(p+1)/2$ parameters. If we have 50 predictors this is some multiple of $(50 * 51)/2 = 1275$ for a significant jump in predictors. The LDA model assumes the K classes share a common covariance matrix so that the LDA model becomes linear in x so that there are kp linear coefficients to estimate. Thus LDA is a less flexible classifer than QDA and has a significantly lower variance. The tradeoff comes from noting that if the LDA assumption of a common covariance matrix is incorrect then high bias can be an issue.

Build the QDA model to evaluate the PCA Test and Train Data

We build the QDA model using the *qda()* function with 'label' as the response variable and using the training data set 'train_data' as the predictor set. Using the input created for the LDA we have y_t to be matched to the first column to be equal to 0 and then building the training and test sets we can build the QDA to recognize "0" and output the accuracy using *mean()* function applied to the predictions generated by the *predict()* function on the model using the test data.

```
######## very impressive result!!!

m_qda <- qda(label ~ .,data = train_data)

qda.pred <- predict(m_qda , test_mnist)

mean(qda.pred$class == y_test)
```

```
## [1] 0.9933125
```

Results

After running our LDA and QDA models on PCA adjusted data we have excellent results with 98% and 99% accuracy respectively.