Principal Component Analysis and LDA, QDA, LR, KNN models

Chris Schmidt

10/24/2019

PCA and modeling with LDA, QDA, LR, and KNN models

This builds on a similar project but we run principal component analysis on the data and add a K-nearest neighbors classifier to the mix of models that are evaluated.

Using the data from problem 11 in HW7, perform PCA for the training data then use the methods in the following list to train classification models and make predictions for test data. Use logistic regression, lda, qda, and knn with k = 3.

(a) Create variable mpg01 and data frame hold the data and new variable.

```
mpg01 <- (Auto$mpg > median(Auto$mpg))*1
AutoData <- data.frame(mpg01, Auto)
dim(AutoData)
```

```
## [1] 392  10
```

(b) Split the data into a training set and a test set.

This is an effort at using a different approach than using createDataPartition().

```
#train_Auto <- AutoData[1:276,1:9]
#test_Auto <- AutoData[277:392,1:9]
#test_data = data.frame(test_Auto)
```

```
set.seed(1)
inTraining <- createDataPartition(AutoData$mpg01, p = 0.7, list = FALSE)
training <- AutoData[inTraining, ]
testing  <- AutoData[-inTraining, ]
dim(training)
```

```
## [1] 276  10
```

Principal Component Analysis

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.

```
pca_train <- (AutoData[,3:8])[inTraining,]
pca_test <- (AutoData[,3:8])[-inTraining,]
dim(pca_train)
```

create training and test sets for pca application

```
## [1] 276   6
```

```
dim(pca_test)
```

```
## [1] 116   6
```

```
#pca_train <- (train_Auto[,3:9]) #tried a different approach here - got the same output
#pca_test <- (test_Auto[,3:9]) #tried a different approach......
```

```
y_t <- (AutoData[, 1])
str(y_t)
```

```
##  num [1:392] 0 0 0 0 0 0 0 0 0 0 ...
```

```
y_train <- y_t[1:276]
y_test <- y_t[277:392]
```

(c) Apply PCA to the training data using prcomp().

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

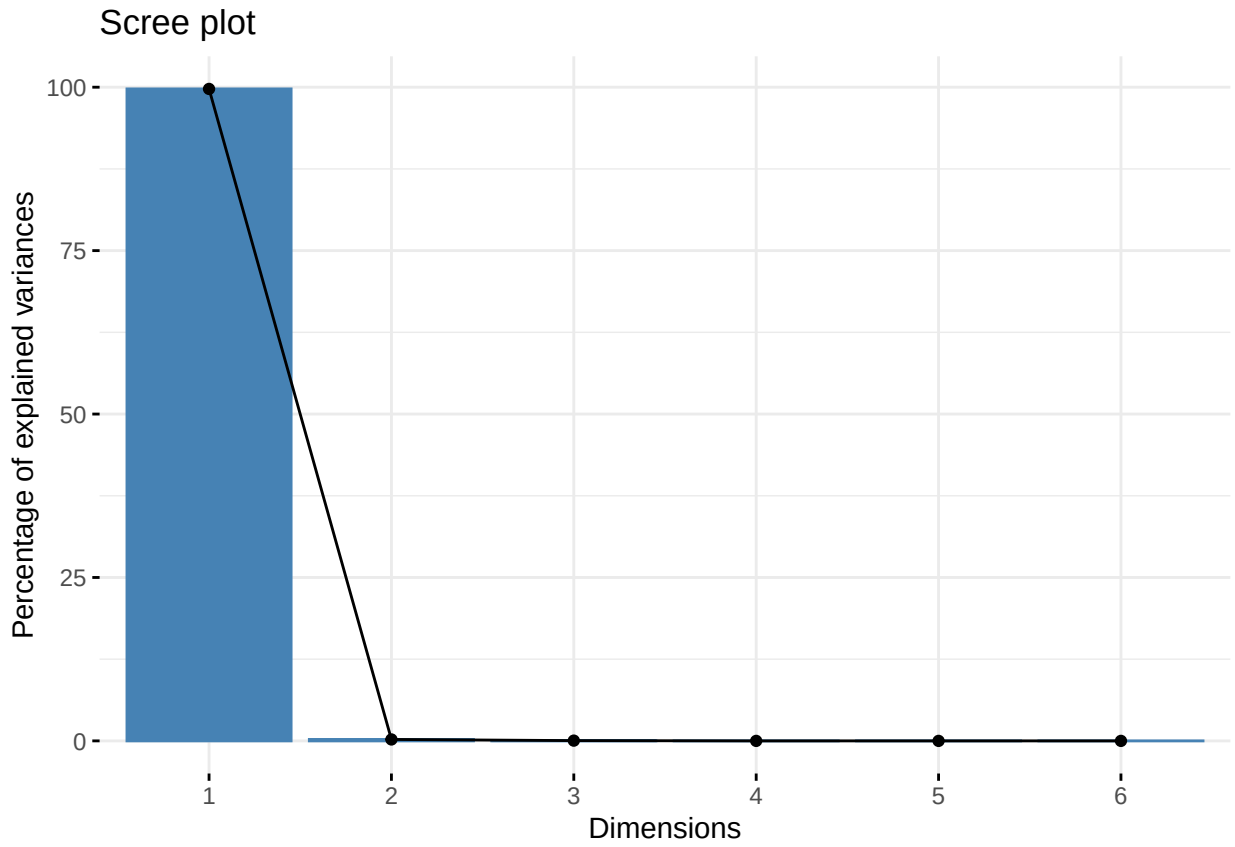
```
pca_training <- prcomp(pca_train, scale. = F)
dim(pca_training$x)
```

```
## [1] 276   6
```

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

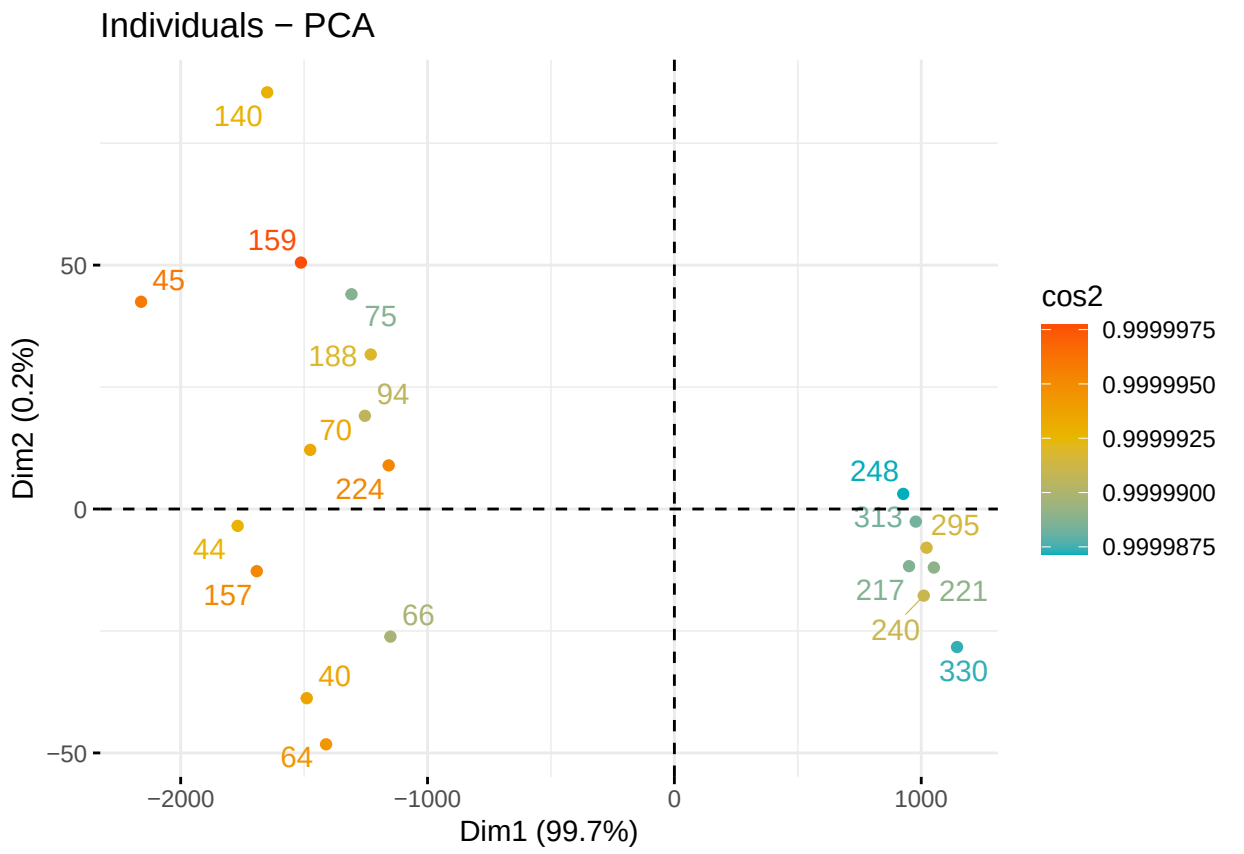
```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

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




```
#biplot(pca_training, scale = 0)
```

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



2. Create the test set Use data.frame to create training set with label.

```
training_pca <- data.frame(label = y_train, pca_training$x[1:276,1:6])
dim(training_pca)
```

```
## [1] 276   7
```

```
cm <- colMeans(pca_train[1:276, ])
test_AutoData <- sweep(pca_test, 2, cm, '-')
test_AutoData <- as.matrix(test_AutoData)%*%pca_training$rotation
colnames(test_AutoData)
```

```
## [1] "PC1" "PC2" "PC3" "PC4" "PC5" "PC6"
```

```
test_AutoData <- data.frame(label = y_test, test_AutoData[, 1:6])
```

(d) Perform LDA on the training data after PCA is applied and find the prediction error.

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 π_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.

#####(The output values seem to be off but I'm not sure)

```
lda_m1 <- lda(label ~ PC1 + PC2 , data = training_pca)
lda.pred <- predict(lda_m1, test_AutoData)
mean(lda.pred$class != y_test) #error
```

```
## [1] 0.8448276
```

```
mean(lda.pred$class == y_test) #prediction success rate
```

```
## [1] 0.1551724
```

```
#table(lda.pred, test_AutoData$label)
#error_lda <- mean(lda.pred != test_AutoData$label)
#error_lda
```

Tried to emulate what I used on HW7 but it didn't work.

(e) Perform QDA on the training data after PCA is applied and find the prediction error.

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.

```
qda_m1 <- qda(label ~ PC1 + PC2 + PC3 + PC4, data = training_pca)
qda.pred <- predict(qda_m1, test_AutoData)
mean(qda.pred$class != y_test)
```

```
## [1] 0.7758621
```

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

```
## [1] 0.2241379
```

(f) Perform logistic regression on the training data after PCA is applied and find the prediction error.

```
logisticR_m1 <-glm(label ~ PC1 + PC2, data = training_pca, family = binomial)
logisticR.prob <- predict(logisticR_m1, test_AutoData, type = "response")
logisticR.pred <- ifelse(logisticR.prob > 0.5, 1, 0)
mean(logisticR.pred != y_test)
```

```
## [1] 0.8448276
```

```
mean(logisticR.pred == y_test)
```

```
## [1] 0.1551724
```

(g) Perform KNN on the training data after PCA is applied, with k = 3, and find the Prediction error.

```
training.X <- cbind(training_pca$PC1, training_pca$PC2, training_pca$PC3, training_pca$PC4)
testing.X <- cbind(test_AutoData$PC1, test_AutoData$PC2, test_AutoData$PC3, test_AutoData$PC4 )
knn.pred <- knn(training.X, testing.X, training$mpg01, k = 3)
mean(knn.pred != y_test)
```

```
## [1] 0.4396552
```

```
mean(knn.pred == y_test)
```

```
## [1] 0.5603448
```