Lab 7

Adeline Casali

2024-02-29

Principal Components Analysis

# Loading packages  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(reticulate)  
library(tensorflow)

##   
## Attaching package: 'tensorflow'

## The following object is masked from 'package:caret':  
##   
## train

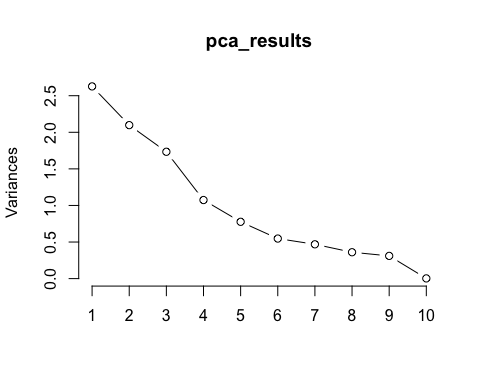
library(keras)  
library(MESS)  
  
# Loading data and preprocessing  
data <- read.csv("lab\_7\_data.csv")  
training\_ind <- createDataPartition(data$lodgepole\_pine,  
 p = 0.75,  
 list = FALSE,  
 times = 1)  
training\_set <- data[training\_ind, ]  
test\_set <- data[-training\_ind, ]  
top\_20\_soil\_types <- training\_set %>%  
 group\_by(soil\_type) %>%  
 summarise(count = n()) %>%  
 arrange(desc(count)) %>%  
 select(soil\_type) %>%  
 top\_n(20)

## Selecting by soil\_type

training\_set$soil\_type <- ifelse(training\_set$soil\_type %in% top\_20\_soil\_types$soil\_type,  
 training\_set$soil\_type,  
 "other")  
training\_set$wilderness\_area <- factor(training\_set$wilderness\_area)  
training\_set$soil\_type <- factor(training\_set$soil\_type)  
onehot\_encoder <- dummyVars(~ wilderness\_area + soil\_type,  
 training\_set[, c("wilderness\_area", "soil\_type")],  
 levelsOnly = TRUE,  
 fullRank = TRUE)  
onehot\_enc\_training <- predict(onehot\_encoder,  
 training\_set[, c("wilderness\_area", "soil\_type")])  
training\_set <- cbind(training\_set, onehot\_enc\_training)  
test\_set$soil\_type <- ifelse(test\_set$soil\_type %in% top\_20\_soil\_types$soil\_type,  
 test\_set$soil\_type,  
 "other")  
test\_set$wilderness\_area <- factor(test\_set$wilderness\_area)  
test\_set$soil\_type <- factor(test\_set$soil\_type)  
onehot\_enc\_test <- predict(onehot\_encoder, test\_set[, c("wilderness\_area", "soil\_type")])  
test\_set <- cbind(test\_set, onehot\_enc\_test)  
test\_set[, -c(11:13)] <- scale(test\_set[, -c(11:13)],  
 center = apply(training\_set[, -c(11:13)], 2, mean),  
 scale = apply(training\_set[, -c(11:13)], 2, sd))  
training\_set[, -c(11:13)] <- scale(training\_set[, -c(11:13)])  
training\_features <- array(data = unlist(training\_set[, -c(11:13)]),  
 dim = c(nrow(training\_set), 33))  
training\_labels <- array(data = unlist(training\_set[, 13]),  
 dim = c(nrow(training\_set)))  
test\_features <- array(data = unlist(test\_set[, -c(11:13)]),  
 dim = c(nrow(test\_set), 33))  
test\_labels <- array(data = unlist(test\_set[, 13]),  
 dim = c(nrow(test\_set)))  
  
# Running PCA  
pca\_results <- prcomp(training\_features[, 1:10])  
summary(pca\_results)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 1.6209 1.4484 1.3167 1.0364 0.88117 0.73989 0.6848  
## Proportion of Variance 0.2627 0.2098 0.1734 0.1074 0.07765 0.05474 0.0469  
## Cumulative Proportion 0.2627 0.4725 0.6459 0.7533 0.83095 0.88570 0.9326  
## PC8 PC9 PC10  
## Standard deviation 0.60053 0.55722 0.05392  
## Proportion of Variance 0.03606 0.03105 0.00029  
## Cumulative Proportion 0.96866 0.99971 1.00000

screeplot(pca\_results, type = "line")



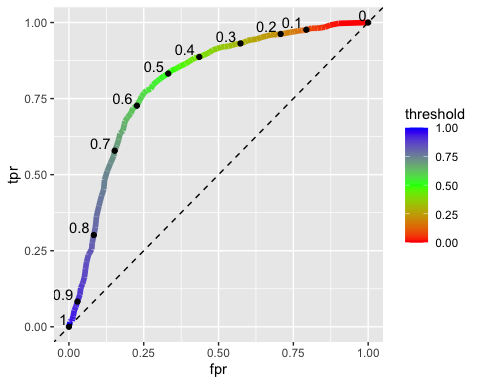
# Reducing to 6 PCs  
training\_rotated <- as.matrix(training\_features[, 1:10] %\*% pca\_results$rotation)  
training\_features <- cbind(training\_features, training\_rotated[, 1:6])  
  
test\_rotated <- as.matrix(test\_features[, 1:10]) %\*% pca\_results$rotation  
test\_features <- cbind(test\_features, test\_rotated[, 1:6])  
  
# Creating a neural network model with PCA features  
model <- keras\_model\_sequential(list(  
 layer\_dense(units = 50, activation = "relu"),  
 layer\_dense(units = 25, activation = "relu"),  
 layer\_dense(units = 1, activation = "sigmoid")  
))  
compile(model,  
 optimizer = "rmsprop",  
 loss = "binary\_crossentropy",  
 metrics = "accuracy")  
history <- fit(model, training\_features, training\_labels,  
 epochs = 40, batch\_size = 512, validation\_split = 0.33)

## Epoch 1/40  
## 9/9 - 1s - loss: 0.7655 - accuracy: 0.5248 - val\_loss: 0.7058 - val\_accuracy: 0.5728 - 1s/epoch - 119ms/step  
## Epoch 2/40  
## 9/9 - 0s - loss: 0.6253 - accuracy: 0.6712 - val\_loss: 0.6872 - val\_accuracy: 0.5871 - 128ms/epoch - 14ms/step  
## Epoch 3/40  
## 9/9 - 0s - loss: 0.5752 - accuracy: 0.7333 - val\_loss: 0.6899 - val\_accuracy: 0.5894 - 115ms/epoch - 13ms/step  
## Epoch 4/40  
## 9/9 - 0s - loss: 0.5533 - accuracy: 0.7566 - val\_loss: 0.6926 - val\_accuracy: 0.5955 - 113ms/epoch - 13ms/step  
## Epoch 5/40  
## 9/9 - 0s - loss: 0.5395 - accuracy: 0.7634 - val\_loss: 0.7004 - val\_accuracy: 0.6001 - 113ms/epoch - 13ms/step  
## Epoch 6/40  
## 9/9 - 0s - loss: 0.5310 - accuracy: 0.7662 - val\_loss: 0.7089 - val\_accuracy: 0.6182 - 112ms/epoch - 12ms/step  
## Epoch 7/40  
## 9/9 - 0s - loss: 0.5246 - accuracy: 0.7712 - val\_loss: 0.7076 - val\_accuracy: 0.6274 - 113ms/epoch - 13ms/step  
## Epoch 8/40  
## 9/9 - 0s - loss: 0.5205 - accuracy: 0.7723 - val\_loss: 0.7088 - val\_accuracy: 0.6339 - 112ms/epoch - 12ms/step  
## Epoch 9/40  
## 9/9 - 0s - loss: 0.5172 - accuracy: 0.7737 - val\_loss: 0.7037 - val\_accuracy: 0.6390 - 112ms/epoch - 12ms/step  
## Epoch 10/40  
## 9/9 - 0s - loss: 0.5151 - accuracy: 0.7748 - val\_loss: 0.7023 - val\_accuracy: 0.6446 - 112ms/epoch - 12ms/step  
## Epoch 11/40  
## 9/9 - 0s - loss: 0.5132 - accuracy: 0.7769 - val\_loss: 0.7050 - val\_accuracy: 0.6395 - 116ms/epoch - 13ms/step  
## Epoch 12/40  
## 9/9 - 0s - loss: 0.5104 - accuracy: 0.7767 - val\_loss: 0.6962 - val\_accuracy: 0.6627 - 113ms/epoch - 13ms/step  
## Epoch 13/40  
## 9/9 - 0s - loss: 0.5093 - accuracy: 0.7789 - val\_loss: 0.6994 - val\_accuracy: 0.6594 - 112ms/epoch - 12ms/step  
## Epoch 14/40  
## 9/9 - 0s - loss: 0.5074 - accuracy: 0.7776 - val\_loss: 0.7012 - val\_accuracy: 0.6487 - 119ms/epoch - 13ms/step  
## Epoch 15/40  
## 9/9 - 0s - loss: 0.5078 - accuracy: 0.7799 - val\_loss: 0.6961 - val\_accuracy: 0.6654 - 112ms/epoch - 12ms/step  
## Epoch 16/40  
## 9/9 - 0s - loss: 0.5057 - accuracy: 0.7780 - val\_loss: 0.7012 - val\_accuracy: 0.6617 - 112ms/epoch - 12ms/step  
## Epoch 17/40  
## 9/9 - 0s - loss: 0.5066 - accuracy: 0.7769 - val\_loss: 0.6971 - val\_accuracy: 0.6613 - 112ms/epoch - 12ms/step  
## Epoch 18/40  
## 9/9 - 0s - loss: 0.5059 - accuracy: 0.7783 - val\_loss: 0.6927 - val\_accuracy: 0.6701 - 113ms/epoch - 13ms/step  
## Epoch 19/40  
## 9/9 - 0s - loss: 0.5060 - accuracy: 0.7783 - val\_loss: 0.6862 - val\_accuracy: 0.6682 - 111ms/epoch - 12ms/step  
## Epoch 20/40  
## 9/9 - 0s - loss: 0.5056 - accuracy: 0.7778 - val\_loss: 0.6860 - val\_accuracy: 0.6756 - 113ms/epoch - 13ms/step  
## Epoch 21/40  
## 9/9 - 0s - loss: 0.5037 - accuracy: 0.7755 - val\_loss: 0.6845 - val\_accuracy: 0.6793 - 112ms/epoch - 12ms/step  
## Epoch 22/40  
## 9/9 - 0s - loss: 0.5033 - accuracy: 0.7785 - val\_loss: 0.6855 - val\_accuracy: 0.6719 - 111ms/epoch - 12ms/step  
## Epoch 23/40  
## 9/9 - 0s - loss: 0.5042 - accuracy: 0.7767 - val\_loss: 0.6860 - val\_accuracy: 0.6752 - 111ms/epoch - 12ms/step  
## Epoch 24/40  
## 9/9 - 0s - loss: 0.5047 - accuracy: 0.7787 - val\_loss: 0.6780 - val\_accuracy: 0.6664 - 111ms/epoch - 12ms/step  
## Epoch 25/40  
## 9/9 - 0s - loss: 0.5041 - accuracy: 0.7739 - val\_loss: 0.6877 - val\_accuracy: 0.6761 - 112ms/epoch - 12ms/step  
## Epoch 26/40  
## 9/9 - 0s - loss: 0.5035 - accuracy: 0.7789 - val\_loss: 0.6731 - val\_accuracy: 0.6789 - 113ms/epoch - 13ms/step  
## Epoch 27/40  
## 9/9 - 0s - loss: 0.5027 - accuracy: 0.7792 - val\_loss: 0.6766 - val\_accuracy: 0.6849 - 112ms/epoch - 12ms/step  
## Epoch 28/40  
## 9/9 - 0s - loss: 0.5036 - accuracy: 0.7776 - val\_loss: 0.6719 - val\_accuracy: 0.6863 - 111ms/epoch - 12ms/step  
## Epoch 29/40  
## 9/9 - 0s - loss: 0.5028 - accuracy: 0.7796 - val\_loss: 0.6691 - val\_accuracy: 0.6747 - 111ms/epoch - 12ms/step  
## Epoch 30/40  
## 9/9 - 0s - loss: 0.5038 - accuracy: 0.7773 - val\_loss: 0.6748 - val\_accuracy: 0.6826 - 111ms/epoch - 12ms/step  
## Epoch 31/40  
## 9/9 - 0s - loss: 0.5050 - accuracy: 0.7753 - val\_loss: 0.6756 - val\_accuracy: 0.6789 - 111ms/epoch - 12ms/step  
## Epoch 32/40  
## 9/9 - 0s - loss: 0.5028 - accuracy: 0.7794 - val\_loss: 0.6764 - val\_accuracy: 0.6867 - 111ms/epoch - 12ms/step  
## Epoch 33/40  
## 9/9 - 0s - loss: 0.5047 - accuracy: 0.7764 - val\_loss: 0.6774 - val\_accuracy: 0.6766 - 111ms/epoch - 12ms/step  
## Epoch 34/40  
## 9/9 - 0s - loss: 0.5048 - accuracy: 0.7755 - val\_loss: 0.6727 - val\_accuracy: 0.6756 - 113ms/epoch - 13ms/step  
## Epoch 35/40  
## 9/9 - 0s - loss: 0.5027 - accuracy: 0.7785 - val\_loss: 0.6688 - val\_accuracy: 0.6654 - 118ms/epoch - 13ms/step  
## Epoch 36/40  
## 9/9 - 0s - loss: 0.5036 - accuracy: 0.7760 - val\_loss: 0.6669 - val\_accuracy: 0.6747 - 140ms/epoch - 16ms/step  
## Epoch 37/40  
## 9/9 - 0s - loss: 0.5037 - accuracy: 0.7728 - val\_loss: 0.6736 - val\_accuracy: 0.6784 - 131ms/epoch - 15ms/step  
## Epoch 38/40  
## 9/9 - 0s - loss: 0.5054 - accuracy: 0.7764 - val\_loss: 0.6758 - val\_accuracy: 0.6895 - 113ms/epoch - 13ms/step  
## Epoch 39/40  
## 9/9 - 0s - loss: 0.5040 - accuracy: 0.7785 - val\_loss: 0.6673 - val\_accuracy: 0.6770 - 112ms/epoch - 12ms/step  
## Epoch 40/40  
## 9/9 - 0s - loss: 0.5044 - accuracy: 0.7739 - val\_loss: 0.6641 - val\_accuracy: 0.6770 - 112ms/epoch - 12ms/step

predictions <- predict(model, test\_features)

## 69/69 - 0s - 149ms/epoch - 2ms/step

test\_set$p\_prob <- predictions[, 1]  
# ROC curve  
roc\_data <- data.frame(threshold=seq(1,0,-0.01), fpr=0, tpr=0)  
for (i in roc\_data$threshold) {  
 over\_threshold <- test\_set[test\_set$p\_prob >= i, ]  
 fpr <- sum(over\_threshold$lodgepole\_pine==0)/sum(test\_set$lodgepole\_pine==0)  
 roc\_data[roc\_data$threshold==i, "fpr"] <- fpr  
 tpr <- sum(over\_threshold$lodgepole\_pine==1)/sum(test\_set$lodgepole\_pine==1)  
 roc\_data[roc\_data$threshold==i, "tpr"] <- tpr  
}  
ggplot() +  
 geom\_line(data = roc\_data, aes(x = fpr, y = tpr, color = threshold), linewidth = 2) +  
 scale\_color\_gradientn(colors = rainbow(3)) +  
 geom\_abline(intercept = 0, slope = 1, lty = 2) +  
 geom\_point(data = roc\_data[seq(1, 101, 10), ], aes(x = fpr, y = tpr)) +  
 geom\_text(data = roc\_data[seq(1, 101, 10), ],  
 aes(x = fpr, y = tpr, label = threshold, hjust = 1.2, vjust = -0.2))



# AUC  
auc <- auc(x = roc\_data$fpr, y = roc\_data$tpr, type = "spline")

## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique  
## 'x' values

auc

## [1] 0.8066544

1. The main difference between supervised and unsupervised learning is that with supervised learning each observation is associated with a label, and with unsupervised learning there are no labels associated with observations. Because of this, unsupervised learning is often useful for data visualization and dimension reduction, as it works to infer relationships between the observations and between features.
2. Centering is always required for PCA, as PCs are unit vectors that originate from the origin. Scaling is almost always required, in order to ensure that each numerical feature has equal weight. However, it may not be required in situations where all numerical features are measured on similar scales (ie genes) or if you want features with larger variances to have more importance.
3. The 9th feature (0.58907933) has the most influence on the first PC, while the 10th feature (-0.03661076) has the least influence.

pca\_results$rotation

## PC1 PC2 PC3 PC4 PC5  
## [1,] 0.13623182 0.343886098 0.35860347 -0.04062804 -0.615994045  
## [2,] 0.48632185 -0.153675211 -0.09842642 -0.06953254 -0.036307501  
## [3,] -0.08897274 -0.531520368 0.11640698 -0.38058830 -0.148956663  
## [4,] 0.11473799 0.003491365 0.65021988 0.21330483 0.168716442  
## [5,] 0.08995205 -0.233809167 0.60551339 0.12220477 0.210303217  
## [6,] 0.12258721 0.387134106 0.13934932 -0.53610888 -0.273239696  
## [7,] -0.45638734 0.346859723 0.04331907 0.28136390 0.008152849  
## [8,] 0.37958898 0.355959668 -0.12053422 0.36151875 0.147416704  
## [9,] 0.58907933 -0.017259868 -0.11484084 0.01218974 0.093380613  
## [10,] -0.03661076 0.351772357 0.09859595 -0.53926933 0.647436722  
## PC6 PC7 PC8 PC9 PC10  
## [1,] 0.548476392 0.07815316 0.18874220 -0.104506281 2.837106e-03  
## [2,] 0.042331366 0.65362955 -0.51181539 -0.181812222 -6.622998e-05  
## [3,] -0.033901363 0.37007186 0.43073807 0.433586202 -1.317159e-01  
## [4,] 0.007070617 -0.10089689 -0.38971696 0.572679388 1.322799e-03  
## [5,] -0.247930487 0.06302584 0.28263076 -0.605071877 3.002296e-04  
## [6,] -0.660136655 -0.08360367 -0.10238363 0.018795489 -1.735620e-03  
## [7,] -0.157315941 0.45554264 -0.01769521 0.004479634 -5.980716e-01  
## [8,] -0.207464236 0.29848178 0.48683427 0.266187711 3.496465e-01  
## [9,] 0.041112466 -0.30525410 0.17559275 0.047263399 -7.090090e-01  
## [10,] 0.359795121 0.13702101 0.08056489 -0.022114215 1.397321e-03