PCA in Classification problems

Duc Do

2024-11-29

This notebook will discuss and compare several classification techniques for the ***Wisconsin Diagnostic Breast Cancer (WDBC) dataset***, which is a binary classification problem, including the use of Principle Component Analysis. The data dictionary can be accessed ***here*** and more information about the dataset is ***here***.

This notebook is inspired by material from the course Mathematics of Data Science, taught by Prof. Julien Arino at the University of Manitoba.

1. Use `neuralnet` library and 30 attributes.

Install and load `neuralnet`.

```
if (!require("neuralnet")) {
  install.packages("neuralnet")
}
```

```
## Loading required package: neuralnet
```

```
## Warning: package 'neuralnet' was built under R version 4.4.2
```

```
library("neuralnet")
```

Create `col_names` vector to store all the column names as described in the data dictionary.

```
col_names = c("ID", "Diagnosis", "Radius_Mean", "Texture_Mean", "Perimeter_Mean", "Area_Mean", "Smoothn
```

Load the dataset and add column names.

```
dataset = read.csv("https://raw.githubusercontent.com/julien-arino/math-of-data-science/refs/heads/main/
head(dataset)
```

```
##         ID Diagnosis Radius_Mean Texture_Mean Perimeter_Mean Area_Mean
## 1   842302         M       17.99        10.38         122.80    1001.0
## 2   842517         M       20.57        17.77         132.90    1326.0
## 3 84300903         M       19.69        21.25         130.00    1203.0
## 4 84348301         M       11.42        20.38          77.58     386.1
## 5 84358402         M       20.29        14.34         135.10    1297.0
## 6   843786         M       12.45        15.70          82.57     477.1
##   Smoothness_Mean Compactness_Mean Concavity_Mean Concave_Points_Mean
```

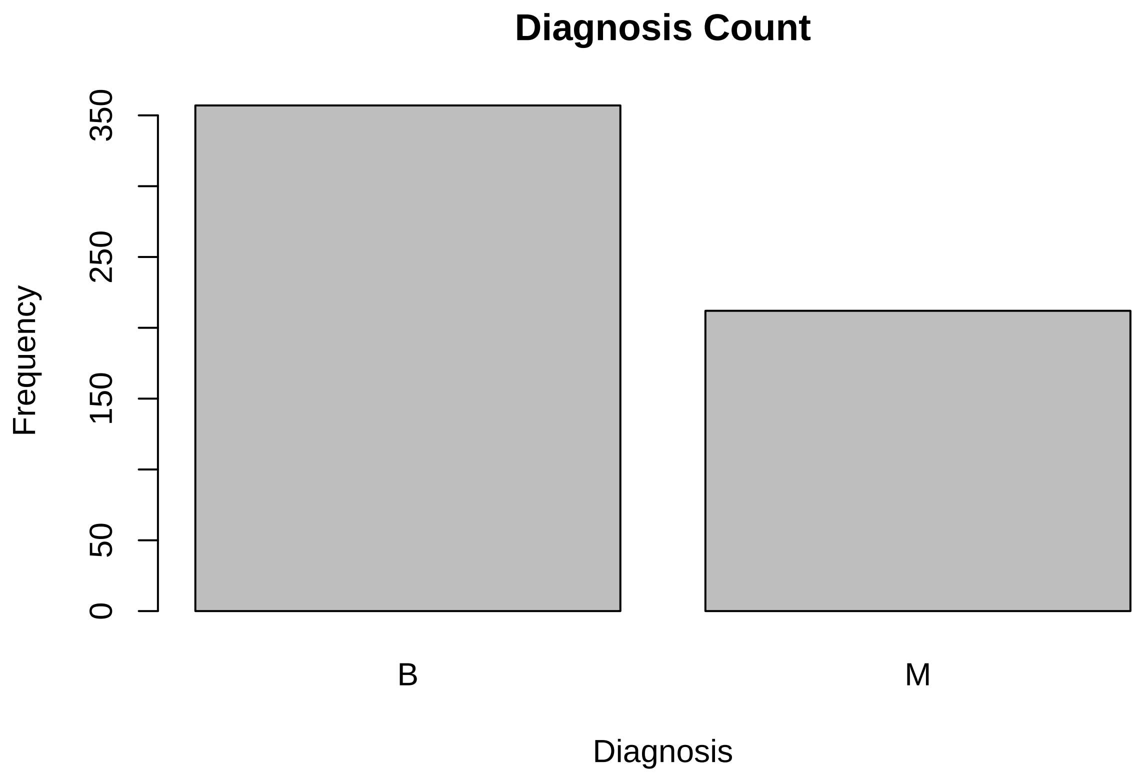
```
## 1         0.11840          0.27760          0.3001             0.14710
## 2         0.08474          0.07864          0.0869             0.07017
## 3         0.10960          0.15990          0.1974             0.12790
## 4         0.14250          0.28390          0.2414             0.10520
## 5         0.10030          0.13280          0.1980             0.10430
## 6         0.12780          0.17000          0.1578             0.08089
##   Symmetry_Mean Fractal_Dimension_Mean Radius_SE Texture_SE Perimeter_SE
## 1        0.2419                0.07871    1.0950     0.9053        8.589
## 2        0.1812                0.05667    0.5435     0.7339        3.398
## 3        0.2069                0.05999    0.7456     0.7869        4.585
## 4        0.2597                0.09744    0.4956     1.1560        3.445
## 5        0.1809                0.05883    0.7572     0.7813        5.438
## 6        0.2087                0.07613    0.3345     0.8902        2.217
##   Area_SE Smoothness_SE Compactness_SE Concavity_SE Concave_Points_SE
## 1  153.40      0.006399        0.04904      0.05373           0.01587
## 2   74.08      0.005225        0.01308      0.01860           0.01340
## 3   94.03      0.006150        0.04006      0.03832           0.02058
## 4   27.23      0.009110        0.07458      0.05661           0.01867
## 5   94.44      0.011490        0.02461      0.05688           0.01885
## 6   27.19      0.007510        0.03345      0.03672           0.01137
##   Symmetry_SE Fractal_Dimension_SE Radius_Worst Texture_Worst Perimeter_Worst
## 1     0.03003             0.006193        25.38         17.33          184.60
## 2     0.01389             0.003532        24.99         23.41          158.80
## 3     0.02250             0.004571        23.57         25.53          152.50
## 4     0.05963             0.009208        14.91         26.50           98.87
## 5     0.01756             0.005115        22.54         16.67          152.20
## 6     0.02165             0.005082        15.47         23.75          103.40
##   Area_Worst Smoothness_Worst Compactness_Worst Concavity_Worst
## 1     2019.0           0.1622            0.6656          0.7119
## 2     1956.0           0.1238            0.1866          0.2416
## 3     1709.0           0.1444            0.4245          0.4504
## 4      567.7           0.2098            0.8663          0.6869
## 5     1575.0           0.1374            0.2050          0.4000
## 6      741.6           0.1791            0.5249          0.5355
##   Concave_Points_Worst Symmetry_Worst Fractal_Dimension_Worst
## 1               0.2654         0.4601                 0.11890
## 2               0.1860         0.2750                 0.08902
## 3               0.2430         0.3613                 0.08758
## 4               0.2575         0.6638                 0.17300
## 5               0.1625         0.2364                 0.07678
## 6               0.1741         0.3985                 0.12440
```

Take a look at `Diagnosis` column which is the label.

```r
table(dataset$Diagnosis)
```

```
##
##   B   M
## 357 212
```

```r
barplot(table(dataset$Diagnosis), main = "Diagnosis Count", xlab = "Diagnosis", ylab = "Frequency")
```



There are 569 observations with 357 belong to class `B` (benign) and 212 belong to class `M` (malignant). This is a binary classification problem where the primary objective is to avoid missing any malignant (`M`) patient, as the cost of a false negative could cost a life. I will discuss the metric for this later.

The dataset is a little bit imbalance. We may handle that later if the result is not good.

Remove the `ID` column (first column) as it is not relevant for classification. As described there is no `Nan` values so no further data pre processing is needed.

```r
data = dataset[, -1]
head(data)
```

```
##   Diagnosis Radius_Mean Texture_Mean Perimeter_Mean Area_Mean Smoothness_Mean
## 1         M       17.99        10.38         122.80    1001.0         0.11840
## 2         M       20.57        17.77         132.90    1326.0         0.08474
## 3         M       19.69        21.25         130.00    1203.0         0.10960
## 4         M       11.42        20.38          77.58     386.1         0.14250
## 5         M       20.29        14.34         135.10    1297.0         0.10030
## 6         M       12.45        15.70          82.57     477.1         0.12780
##   Compactness_Mean Concavity_Mean Concave_Points_Mean Symmetry_Mean
## 1          0.27760         0.3001             0.14710        0.2419
## 2          0.07864         0.0869             0.07017        0.1812
## 3          0.15990         0.1974             0.12790        0.2069
## 4          0.28390         0.2414             0.10520        0.2597
## 5          0.13280         0.1980             0.10430        0.1809
## 6          0.17000         0.1578             0.08089        0.2087
##   Fractal_Dimension_Mean Radius_SE Texture_SE Perimeter_SE Area_SE
```

```
## 1                0.07871    1.0950     0.9053         8.589  153.40
## 2                0.05667    0.5435     0.7339         3.398   74.08
## 3                0.05999    0.7456     0.7869         4.585   94.03
## 4                0.09744    0.4956     1.1560         3.445   27.23
## 5                0.05883    0.7572     0.7813         5.438   94.44
## 6                0.07613    0.3345     0.8902         2.217   27.19
##   Smoothness_SE Compactness_SE Concavity_SE Concave_Points_SE Symmetry_SE
## 1      0.006399        0.04904      0.05373           0.01587     0.03003
## 2      0.005225        0.01308      0.01860           0.01340     0.01389
## 3      0.006150        0.04006      0.03832           0.02058     0.02250
## 4      0.009110        0.07458      0.05661           0.01867     0.05963
## 5      0.011490        0.02461      0.05688           0.01885     0.01756
## 6      0.007510        0.03345      0.03672           0.01137     0.02165
##   Fractal_Dimension_SE Radius_Worst Texture_Worst Perimeter_Worst Area_Worst
## 1             0.006193        25.38         17.33          184.60     2019.0
## 2             0.003532        24.99         23.41          158.80     1956.0
## 3             0.004571        23.57         25.53          152.50     1709.0
## 4             0.009208        14.91         26.50           98.87      567.7
## 5             0.005115        22.54         16.67          152.20     1575.0
## 6             0.005082        15.47         23.75          103.40      741.6
##   Smoothness_Worst Compactness_Worst Concavity_Worst Concave_Points_Worst
## 1           0.1622            0.6656          0.7119               0.2654
## 2           0.1238            0.1866          0.2416               0.1860
## 3           0.1444            0.4245          0.4504               0.2430
## 4           0.2098            0.8663          0.6869               0.2575
## 5           0.1374            0.2050          0.4000               0.1625
## 6           0.1791            0.5249          0.5355               0.1741
##   Symmetry_Worst Fractal_Dimension_Worst
## 1         0.4601                 0.11890
## 2         0.2750                 0.08902
## 3         0.3613                 0.08758
## 4         0.6638                 0.17300
## 5         0.2364                 0.07678
## 6         0.3985                 0.12440
```

Normalize the data to `N(0, 1)` to eliminate unit differences for better classification.

$$X = \frac{X - \bar{X}}{\sigma_X}$$

```r
data[, -1] = scale(data[, -1])
head(data)
```

```
##   Diagnosis Radius_Mean Texture_Mean Perimeter_Mean  Area_Mean Smoothness_Mean
## 1         M   1.0960995   -2.0715123      1.2688173  0.9835095       1.5670875
## 2         M   1.8282120   -0.3533215      1.6844726  1.9070303      -0.8262354
## 3         M   1.5784992    0.4557859      1.5651260  1.5575132       0.9413821
## 4         M  -0.7682333    0.2535091     -0.5921661 -0.7637917       3.2806668
## 5         M   1.7487579   -1.1508038      1.7750113  1.8246238       0.2801253
## 6         M  -0.4759559   -0.8346009     -0.3868077 -0.5052059       2.2354545
##   Compactness_Mean Concavity_Mean Concave_Points_Mean Symmetry_Mean
## 1        3.2806281     2.65054179           2.5302489   2.215565542
## 2       -0.4866435    -0.02382489           0.5476623   0.001391139
```

```
## 3        1.0519999     1.36227979          2.0354398   0.938858720
## 4        3.3999174     1.91421287          1.4504311   2.864862154
## 5        0.5388663     1.36980615          1.4272370  -0.009552062
## 6        1.2432416     0.86554001          0.8239307   1.004517928
##   Fractal_Dimension_Mean  Radius_SE Texture_SE Perimeter_SE    Area_SE
## 1              2.2537638  2.4875451 -0.5647681    2.8305403  2.4853907
## 2             -0.8678888  0.4988157 -0.8754733    0.2630955  0.7417493
## 3             -0.3976580  1.2275958 -0.7793976    0.8501802  1.1802975
## 4              4.9066020  0.3260865 -0.1103120    0.2863415 -0.2881246
## 5             -0.5619555  1.2694258 -0.7895490    1.2720701  1.1893103
## 6              1.8883435 -0.2548461 -0.5921406   -0.3210217 -0.2890039
##   Smoothness_SE Compactness_SE Concavity_SE Concave_Points_SE Symmetry_SE
## 1    -0.2138135     1.31570389    0.7233897        0.66023900   1.1477468
## 2    -0.6048187    -0.69231710   -0.4403926        0.25993335  -0.8047423
## 3    -0.2967439     0.81425704    0.2128891        1.42357487   0.2368272
## 4     0.6890953     2.74186785    0.8187979        1.11402678   4.7285198
## 5     1.4817634    -0.04847723    0.8277425        1.14319885  -0.3607748
## 6     0.1562093     0.44515196    0.1598845       -0.06906279   0.1340009
##   Fractal_Dimension_SE Radius_Worst Texture_Worst Perimeter_Worst Area_Worst
## 1           0.90628565    1.8850310   -1.35809849       2.3015755  1.9994782
## 2          -0.09935632    1.8043398   -0.36887865       1.5337764  1.8888270
## 3           0.29330133    1.5105411   -0.02395331       1.3462906  1.4550043
## 4           2.04571087   -0.2812170    0.13386631      -0.2497196 -0.5495377
## 5           0.49888916    1.2974336   -1.46548091       1.3373627  1.2196511
## 6           0.48641784   -0.1653528   -0.31356043      -0.1149083 -0.2441054
##   Smoothness_Worst Compactness_Worst Concavity_Worst Concave_Points_Worst
## 1        1.3065367         2.6143647       2.1076718            2.2940576
## 2       -0.3752817        -0.4300658      -0.1466200            1.0861286
## 3        0.5269438         1.0819801       0.8542223            1.9532817
## 4        3.3912907         3.8899747       1.9878392            2.1738732
## 5        0.2203623        -0.3131190       0.6126397            0.7286181
## 6        2.0467119         1.7201029       1.2621327            0.9050914
##   Symmetry_Worst Fractal_Dimension_Worst
## 1      2.7482041               1.9353117
## 2     -0.2436753               0.2809428
## 3      1.1512420               0.2012142
## 4      6.0407261               4.9306719
## 5     -0.8675896              -0.3967505
## 6      1.7525273               2.2398308
```

I use 80% of the data for training and 20% for testing.

```r
set.seed(2740)

train_idx = sample(nrow(data), 4/5 * nrow(data))
train = data[train_idx, ]
test = data[-train_idx, ]

nrow(train)
```

```
## [1] 455
```

```r
nrow(test)
```

```
## [1] 114
```

Train the data. We want to predict `M` (malignant) as it is considered dangerous. `B` (benign) in the other hand, is not dangerous. I choose this model with 3 hidden layers of 4, 8 and 4 nodes on each layer respectively, just as an example.

```r
nn_model = neuralnet(Diagnosis == "M" ~ ., data = train, hidden = c(4, 8, 4), linear.output=FALSE)
summary(nn_model)
```

```
##                     Length Class      Mode
## call                    5  -none-     call
## response              455  -none-     logical
## covariate           13650  -none-     numeric
## model.list              2  -none-     list
## err.fct                 1  -none-     function
## act.fct                 1  -none-     function
## linear.output           1  -none-     logical
## data                   31  data.frame list
## exclude                 0  -none-     NULL
## net.result              1  -none-     list
## weights                 1  -none-     list
## generalized.weights     1  -none-     list
## startweights            1  -none-     list
## result.matrix         208  -none-     numeric
```

The number of parameters (weights) of the model is:

```r
num_weights = sum(sapply(nn_model$weights, function(layer) sum(lengths(layer))))
num_weights
```

```
## [1] 205
```

There are 205 weights for this model.

Predict and print out confusion table.

```r
pred = predict(nn_model, newdata = test)

confusion_matrix = table(test$Diagnosis == "M", pred[, 1] > 0.5)
confusion_matrix
```

```
##
##         FALSE TRUE
##   FALSE    67    0
##   TRUE      3   44
```

In the test set of 114 samples, we correctly predicted 44 malignant patients (True Positives) and 67 benign patients (True Negatives). 3 malignant patients is falsely predicted as benign (False Negative). 0 benign patient is falsely predicted as malignant (False Positive).

I make a metrics function which will return `Accuracy`, `Precision`, `Recall` and `F1 score`. For this problem, we want `Recall` to be high as the cost of False Negative is high (failing to identify a cancer patient), while False Positives are less critical (falsely identifying a cancer patient).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = 2\frac{Precision * Recall}{Precision + Recall}$$

```r
# Extract TP, TN, FP, FN
TP = confusion_matrix[2, 2]  # True Positive (Malignant correctly predicted as M)
TN = confusion_matrix[1, 1]  # True Negative (Benign correctly predicted as B)
FP = confusion_matrix[1, 2]  # False Positive (Benign incorrectly predicted as M)
FN = confusion_matrix[2, 1]  # False Negative (Malignant incorrectly predicted as B)

metrics = function(TP, TN, FP, FN) {
  Accuracy = (TP + TN) / (TP + TN + FP + FN)
  Precision = TP / (TP + FP)
  Recall = TP / (TP + FN)
  F1_score = 2 * Precision * Recall / (Precision + Recall)

  return (list(Accuracy = Accuracy, Precision = Precision, Recall = Recall, F1_score = F1_score))
}


metrics(TP, TN, FP, FN)
```

```
## $Accuracy
## [1] 0.9736842
##
## $Precision
## [1] 1
##
## $Recall
## [1] 0.9361702
##
## $F1_score
## [1] 0.967033
```

The model is relatively good with $Recall = 93.61\%$, while other metrics are very high.

To effectively evaluate a model, I create a cross-validation function that takes in the number of folds (k), a vector specifying the hidden layers, and the dataset, and returns the performance metrics of the model.

The model is trained on k-1 folds and tested on the remaining fold. This process is repeated k times, each time using a different fold as the test set. The final performance metric is the average of the metrics from all k iterations.

```r
# perform k-fold cross-validation
cross_validation_nn = function(k, hidden_layers, data) {
  # create k-folds
  folds = sample(1:k, nrow(data), replace = TRUE)

  accuracies = numeric(k)
  precisions = numeric(k)
  recalls = numeric(k)
  f1_scores = numeric(k)

  # loop through each fold
  for (i in 1:k) {
    # split data into training and test sets
    test_idx = which(folds == i)
    train_fold = train[-test_idx, ]
    test_fold = train[test_idx, ]

    # train the model
    nn_model = neuralnet(Diagnosis == "M" ~ ., data = train_fold, hidden = hidden_layers, linear.output

    # predict on the test set
    pred = predict(nn_model, newdata = test_fold)

    confusion_matrix = table(test_fold$Diagnosis == "M", pred[, 1] > 0.5)

    TP = confusion_matrix[2, 2]
    TN = confusion_matrix[1, 1]
    FP = confusion_matrix[1, 2]
    FN = confusion_matrix[2, 1]

    accuracies[i] = metrics(TP, TN, FP, FN)$Accuracy
    precisions[i] = metrics(TP, TN, FP, FN)$Precision
    recalls[i] = metrics(TP, TN, FP, FN)$Recall
    f1_scores[i] = metrics(TP, TN, FP, FN)$F1_score
  }

  return (list(Accuracy = mean(accuracies), Precision = mean(precisions), Recall = mean(recalls), F1_sc
}
```

Try 3 different models 5 times with 5-fold cross validation. We use `Recall` as our metric as explained.

```r
set.seed(2740)
n = 5

mean(sapply(1:n, function(x) cross_validation_nn(5, c(4, 8, 4), data)$Recall))
```

```
## [1] 0.957787
```

```r
mean(sapply(1:n, function(x) cross_validation_nn(5, c(16, 16), data)$Recall))
```

```
## [1] 0.9606575
```

```r
mean(sapply(1:n, function(x) cross_validation_nn(5, c(32), data)$Recall))
```

```
## [1] 0.9607794
```

Seem like the model with 1 hidden layers of 32 nodes perform the best with $Recall = 96.08\%$.

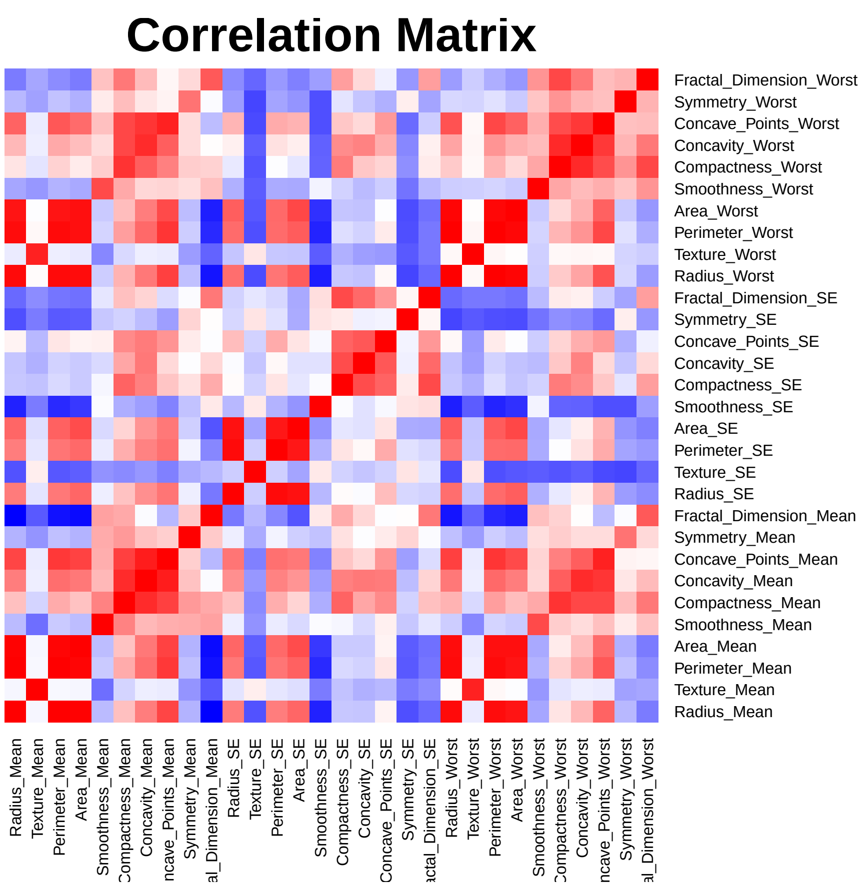
2. Less parameters by using PCA for dimension reduction (use 3 principle components).

a) Perform PCA

First, I compute correlation coefficient matrix for 30 attributes and visualize it (red for high correlation, white for no correlation and blue for low correlation).

```r
# correlation matrix
cor_matrix = cor(data[, -1])

# visualize
heatmap(cor_matrix, main = "Correlation Matrix",
        col = colorRampPalette(c("blue", "white", "red"))(200),
        scale = "none",
        cexRow = 0.6, cexCol = 0.6,
        Rowv = NA, Colv = NA)
```



Some attribute pairs are highly correlated, which may lead to overfitting if both are used. To address this, I will apply PCA to the dataset. It would also help reduce the number of parameters, which means less resources used.

Perform PCA for row reduction (use prcomp).

```
pca_result = prcomp(data[, -1], center = TRUE, scale = TRUE)
summary(pca_result)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     3.6444 2.3857 1.67867 1.40735 1.28403 1.09880 0.82172
## Proportion of Variance 0.4427 0.1897 0.09393 0.06602 0.05496 0.04025 0.02251
## Cumulative Proportion  0.4427 0.6324 0.72636 0.79239 0.84734 0.88759 0.91010
##                            PC8    PC9    PC10   PC11    PC12    PC13    PC14
## Standard deviation     0.69037 0.6457 0.59219 0.5421 0.51104 0.49128 0.39624
## Proportion of Variance 0.01589 0.0139 0.01169 0.0098 0.00871 0.00805 0.00523
## Cumulative Proportion  0.92598 0.9399 0.95157 0.9614 0.97007 0.97812 0.98335
##                           PC15    PC16    PC17    PC18    PC19    PC20   PC21
## Standard deviation     0.30681 0.28260 0.24372 0.22939 0.22244 0.17652 0.1731
## Proportion of Variance 0.00314 0.00266 0.00198 0.00175 0.00165 0.00104 0.0010
## Cumulative Proportion  0.98649 0.98915 0.99113 0.99288 0.99453 0.99557 0.9966
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.16565 0.15602 0.1344 0.12442 0.09043 0.08307 0.03987
## Proportion of Variance 0.00091 0.00081 0.0006 0.00052 0.00027 0.00023 0.00005
## Cumulative Proportion  0.99749 0.99830 0.9989 0.99942 0.99969 0.99992 0.99997
##                           PC29    PC30
## Standard deviation     0.02736 0.01153
## Proportion of Variance 0.00002 0.00000
## Cumulative Proportion  1.00000 1.00000
```

Compute the proportion of variations $= \frac{\sigma_i^2}{\sum_{j=1}^{30} \sigma_j^2}$ where σ_i is the standard deviation of PC_i, obtained from the vector `pca_result$sdev`.

```
prcomp_proportionVariate = pca_result$sdev^2/sum(pca_result$sdev^2)
round(prcomp_proportionVariate, 5)
```

```
##  [1] 0.44272 0.18971 0.09393 0.06602 0.05496 0.04025 0.02251 0.01589 0.01390
## [10] 0.01169 0.00980 0.00871 0.00805 0.00523 0.00314 0.00266 0.00198 0.00175
## [19] 0.00165 0.00104 0.00100 0.00091 0.00081 0.00060 0.00052 0.00027 0.00023
## [28] 0.00005 0.00002 0.00000
```

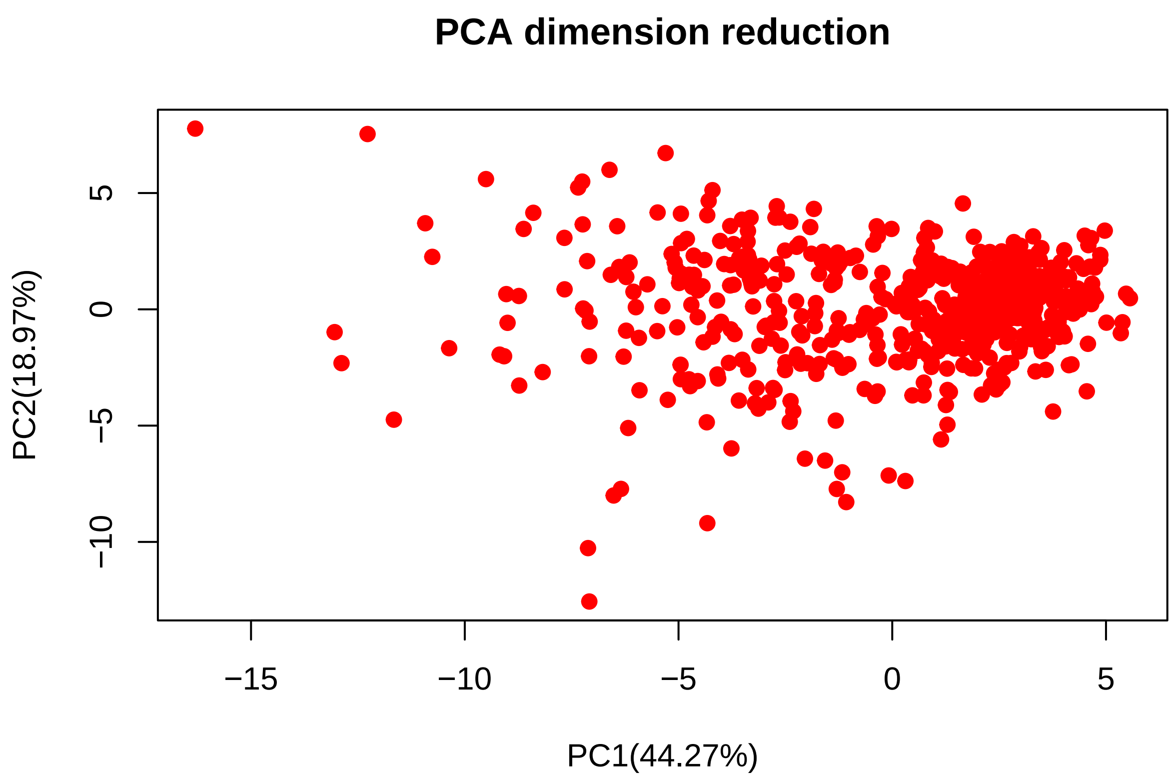
```
sum(round(prcomp_proportionVariate, 5)[1:3])
```

```
## [1] 0.72636
```

I use PC1, PC2 and PC3 for classification. They represent 72.64% variation of this dataset.

Some visualization for PC1 and PC2.

```r
plot(pca_result$x[, 1], pca_result$x[, 2], xlab = "PC1(44.27%)", ylab = "PC2(18.97%)", col = "red", pch
```



Create a new data set with 4 attributes (Diagnosis, PC1, PC2, PC3).

```r
# Create data.PC3 with the first column and the first 3 principal components
data.PC3 = data.frame(Diagnosis = data[, 1], PC1 = pca_result$x[, 1], PC2 = pca_result$x[, 2], PC3 = pca

head(data.PC3)
```

```
##   Diagnosis        PC1        PC2        PC3
## 1         M -9.184755  -1.946870 -1.1221788
## 2         M -2.385703   3.764859 -0.5288274
## 3         M -5.728855   1.074229 -0.5512625
## 4         M -7.116691 -10.266556 -3.2299475
## 5         M -3.931842   1.946359  1.3885450
## 6         M -2.378155  -3.946456 -2.9322967
```

Or some may prefer `PCA` from scratch (without using `prcomp`) to understand it better.

This part is just for reference on how to mathematically perform `PCA` as we would do by hand.

Compute the covariance matrix for this data:

$$S = \frac{1}{n-1} X^T X$$

```r
X = as.matrix(data[, -1])
S = (1/(dim(X)[1]-1)) * t(X) %*% X
```

Compute its eigenvalues. They represent the proportion of variation explained by each principal component.

```r
ev = eigen(S)
```

The proportion of variation for each principal component i is $\frac{e_i}{\Sigma e}$.

```r
proportionVariate = ev$values/(sum(ev$values))
round(proportionVariate, 5)
```

```
##  [1] 0.44272 0.18971 0.09393 0.06602 0.05496 0.04025 0.02251 0.01589 0.01390
## [10] 0.01169 0.00980 0.00871 0.00805 0.00523 0.00314 0.00266 0.00198 0.00175
## [19] 0.00165 0.00104 0.00100 0.00091 0.00081 0.00060 0.00052 0.00027 0.00023
## [28] 0.00005 0.00002 0.00000
```

```r
sum(proportionVariate[1:3])
```

```
## [1] 0.7263637
```

Since the covariance matrix `S` is symmetric, its eigenvectors are orthogonal. The eigenvectors matrix of `S` is our wanted `PCA` basis.

Next, we compute the change of basis `P` from the standard basis to the eigenvector basis. First, create an identity matrix and combine with the eigenvector matrix to get the augmented matrix `A`.

```r
Id = diag(1, nrow =dim(ev$vectors)[1])
A = cbind(ev$vectors, Id)
```

Compute the RREF and extract the relevant change of basis matrix `P`.

$$RREF(eigenvectors(S)|I) = RREF(A) = [I|P]$$

```r
if (!require("pracma")) {
  install.packages("pracma")
}
```

```
## Loading required package: pracma
```

```r
library(pracma)

P = pracma::rref(A)[,(dim(ev$vectors)[2]+1):dim(A)[2]]
```

Finally, compute the new data representation after the rotation. Note that $X_i^{new} = PX_i$ where X_i is a sample from the data, or a row of X. We generalizes this to the entire dataset as $X_{new} = XP^T$.

```r
X_new = X %*% t(P)
head(X_new)
```

```
##           [,1]      [,2]       [,3]      [,4]       [,5]        [,6]
## [1,] -9.184755  1.946870 -1.1221788 3.6305364 -1.1940595  1.41018364
## [2,] -2.385703 -3.764859 -0.5288274 1.1172808  0.6212284  0.02863116
## [3,] -5.728855 -1.074229 -0.5512625 0.9112808 -0.1769302  0.54097615
## [4,] -7.116691 10.266556 -3.2299475 0.1524129 -2.9582754  3.05073750
## [5,] -3.931842 -1.946359  1.3885450 2.9380542  0.5462667 -1.22541641
## [6,] -2.378155  3.946456 -2.9322967 0.9402096 -1.0551135 -0.45064213
##             [,7]        [,8]        [,9]      [,10]      [,11]      [,12]
## [1,]  2.15747152  0.39805698 -0.15698023 -0.8766305  0.2627243 -0.8582593
## [2,]  0.01334635 -0.24077660 -0.71127897  1.1060218  0.8124048  0.1577838
## [3,] -0.66757908 -0.09728813  0.02404449  0.4538760 -0.6050715  0.1242777
## [4,]  1.42865363 -1.05863376 -1.40420412 -1.1159933 -1.1505012  1.0104267
## [5,] -0.93538950 -0.63581661 -0.26357355  0.3773724  0.6507870 -0.1104183
## [6,]  0.49001396  0.16529843 -0.13335576 -0.5299649  0.1096698  0.0813699
##            [,13]        [,14]        [,15]       [,16]       [,17]       [,18]
## [1,]  0.10329677  0.690196797 -0.601264078 -0.74446075  0.26523740  0.54907956
## [2,] -0.94269981  0.652900844  0.008966977  0.64823831  0.01719707 -0.31801756
## [3,] -0.41026561 -0.016665095  0.482994760 -0.32482472 -0.19075064  0.08789759
## [4,] -0.93245070  0.486988399 -0.168699395 -0.05132509 -0.48220960  0.03584323
## [5,]  0.38760691  0.538706543  0.310046684  0.15247165 -0.13302526  0.01869779
## [6,] -0.02625135 -0.003133944  0.178447576  0.01270566 -0.19671335  0.29727706
##           [,19]       [,20]        [,21]       [,22]       [,23]        [,24]
## [1,]  0.1336499  0.34526111 -0.096430045 -0.06878939  0.08444429 -0.175102213
## [2,] -0.2473470 -0.11403274  0.077259494  0.09449530 -0.21752666  0.011280193
## [3,] -0.3922812 -0.20435242 -0.310793246  0.06025601 -0.07422581  0.102671419
## [4,] -0.0267241 -0.46432511 -0.433811661  0.20308706 -0.12399554  0.153294780
## [5,]  0.4610302  0.06543782  0.116442469  0.01763433  0.13933105 -0.005327110
## [6,] -0.1297265 -0.07117453  0.002400178  0.10108043  0.03344819  0.002837749
##             [,25]        [,26]       [,27]         [,28]        [,29]
## [1,] -0.150887294 -0.201326305  0.25236294 -0.0338846387  0.045607590
## [2,] -0.170360355 -0.041092627 -0.18111081  0.0325955021 -0.005682424
## [3,]  0.171007656  0.004731249 -0.04952586  0.0469844833  0.003143131
## [4,]  0.077427574 -0.274982822 -0.18330078  0.0424469831 -0.069233868
## [5,]  0.003059371  0.039219780 -0.03213957 -0.0347556386  0.005033481
## [6,]  0.122282765 -0.030272333  0.08438081  0.0007296587 -0.019703996
##              [,30]
## [1,]  0.0471277407
## [2,]  0.0018662342
## [3,] -0.0007498749
## [4,]  0.0199198881
## [5,] -0.0211951203
## [6,] -0.0034564331
```

Compare the new data representation from scratch and with `prcomp`.

```r
all.equal(abs(pca_result$x), abs(X_new), tolerance = 1e-10, check.class = FALSE, check.attributes = FALS
```

```
## [1] TRUE
```

We see that our manually computed result for X_{new} is correct, which means we can use `X_new` instead of `pca_result$x`.

b) Result of PCA

Similarly, use the `cross_validation_nn` function created before to assess the models (I use less nodes for these model compared to those without PCA).

```
set.seed(2740)

mean(sapply(1:n, function(x) cross_validation_nn(5, c(2, 4, 2), data.PC3)$Recall))
```

```
## [1] 0.9548777
```

```
mean(sapply(1:n, function(x) cross_validation_nn(5, c(8, 8), data.PC3)$Recall))
```

```
## [1] 0.9536758
```

```
mean(sapply(1:n, function(x) cross_validation_nn(5, c(16), data.PC3)$Recall))
```

```
## [1] 0.9670817
```

Similar, the model number 3 with 1 hidden layers with 16 nodes perform the best with $Recall = 96.71\%$, which is slightly better but with significant less parameters compared to without-PCA model.

```
nn_model_1 = neuralnet(Diagnosis == "M" ~ ., data = data, hidden = c(32), linear.output=FALSE)
sum(sapply(nn_model_1$weights, function(layer) sum(lengths(layer))))
```

```
## [1] 1025
```

```
nn_model_2 = neuralnet(Diagnosis == "M" ~ ., data = data.PC3, hidden = c(16), linear.output=FALSE)
sum(sapply(nn_model_2$weights, function(layer) sum(lengths(layer))))
```

```
## [1] 81
```

The best model from part 1 uses 1025 parameters, while the best one in part 2 uses only 81.

3. KNN (K-Nearest Neighbors)

Install and load `class` package for the `knn` function.

```
if (!require("class")) {
  install.packages("class")
}
```

```
## Loading required package: class
```

```
library("class")
```

I use the already computed `train` and `test` sets for `KNN`, with $k = 5$ (meaning that the class of a sample will be determined by its 5 nearest neighbors).

```r
set.seed(2740)

# prepare training and test input
train_labels = train$Diagnosis
test_labels = test$Diagnosis
train_features = train[, -1]
test_features = test[, -1]

# apply KNN
k = 5  # number of neighbors
knn_predictions = knn(train_features, test_features, cl = train_labels, k = k)

# results
confusion_matrix.KNN = table(test_labels, knn_predictions)
rownames(confusion_matrix.KNN) = c("Actual B", "Actual M")
colnames(confusion_matrix.KNN) = c("Predicted B", "Predicted M")

confusion_matrix.KNN
```

```
##            knn_predictions
## test_labels Predicted B Predicted M
##    Actual B          67           0
##    Actual M           3          44
```

```r
TP.KNN = confusion_matrix.KNN[2, 2]
TN.KNN = confusion_matrix.KNN[1, 1]
FP.KNN = confusion_matrix.KNN[1, 2]
FN.KNN = confusion_matrix.KNN[2, 1]


metrics(TP.KNN, TN.KNN, FP.KNN, FN.KNN)
```

```
## $Accuracy
## [1] 0.9736842
##
## $Precision
## [1] 1
##
## $Recall
## [1] 0.9361702
##
## $F1_score
## [1] 0.967033
```

$Recall = 93.62$ is slightly worse than neural net models, but it is still acceptable.

4. Conclusion

The best model is the neural net (1 hidden layer with 16 nodes) with PCA applied on the data from part 2 with $Recall = 96.71\%$, which is quite high compared to the best $accuracy = 97.5\%$ given in the data dictionary.