CW5

Wen Hans Tan and Eirshad Fahim

2025-02-30

Problem 2: Practical (20 marks)

```r
library(readr)
set.seed(42)

#Call Data Set
sonar_data <- read_csv("C:/Users/tanwe/OneDrive/Documents/Stats_Machine_Learning/CW5/sonar.csv")
```

```
## Rows: 208 Columns: 61
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (1): Class
## dbl (60): V1, V2, V3, V4, V5, V6, V7, V8, V9, V10, V11, V12, V13, V14, V15, ...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```r
#Get shuffled indices
indices <- sample(1:nrow(sonar_data),)

#Define train and test split sizes
train_size <- ceiling(0.75 * nrow(sonar_data))
test_size <- nrow(sonar_data) - train_size

#Split data based on exact indices
train_data <- sonar_data[indices[1:train_size], ]
test_data <- sonar_data[indices[(train_size+1):nrow(sonar_data)], ]

#Confirm Training and Testing Data Size
cat("Ful Data:", nrow(sonar_data), "\n")
```

```
## Ful Data: 208
```

```r
cat("Training size:", nrow(train_data), "\n")
```

```
## Training size: 156
```

```r
cat("Testing size:", nrow(test_data), "\n")
```

```
## Testing size: 52
```

```r
#Split features and labels for each dataset
train_features <- train_data[, -ncol(train_data)]
test_features <- test_data[, -ncol(test_data)]
train_labels <- train_data$Class
test_labels <- test_data$Class
```

(i) Write a function **my_gauss_sketch** *which receives as input a pair of natural numbers (r, p) which satisfy $1 \leq r \leq p$, and outputs a matrix S of shape $(r \times p)$ with iid Gaussian entries, with variance chosen so that the expected squared norm of each row is equal to p.*

For every row, each entry is generated independently with distribution $x_{ij} \sim \mathcal{N}(0, \sigma^2)$

Since the expected mean is 0, we get $\mathbb{E}[x_j^2] = \sigma^2$

The squared norm for each row is defined $||x||^2 = \sum_{j=1}^{p} x_j^2$

We want $\mathbb{E}[||x||^2] = p$, but $\mathbb{E}[||x||^2] = p \cdot \mathbb{E}[x_j^2] = p \cdot \sigma^2$. Hence, we choose $\sigma^2 = 1$.

```r
my_gauss_sketch <- function(r,p) {
  if (r < 1 || r > p) {
    stop("Please ensure that 1<=r<=p.")
  }

  S <- matrix(rnorm(r*p, mean = 0 , sd = 1), nrow = r, ncol = p)
  return(S)
}
```

(ii) (5 marks) Write a function my_sse ***which receives as input the triple of natural numbers*** (s, r, p) ***which satisfy*** $1 <= s <= r <= p$, ***and outputs a matrix*** S ***of shape*** $(r\text{Ö}p)$, ***such that:***

1. The rows of S ***are independent,***

2. Each row of S ***contains only*** s ***non-zero entries, and***

3. Each of these non-zero entries is equally likely to be either of $\pm \sqrt{\frac{p}{s}}$

(When using these random embedding matrices in practice, it is recommended to take s = min {r, 8}).

```r
my_sse <- function(r,p, s = NULL){
  #Select s as recommended in practice if not s provided
  s <- min(r,8)

  if (s < 1||r < s||p<r) {
    stop("Please ensure that 1<=r<=p ")
  }
  #Create a r x p matrix filled with zeros
  S <- matrix(0, nrow = r, ncol = p)

  #Looping through r columns
  for (i in 1:r){
    #Randomly select s indices from 1 to p
    indices <- sample(p, s)
    #For each selected position assign value with equal probabiluty
    values <- sample(c(sqrt(p/s),-sqrt(p/s)), s, replace = TRUE)
    S[i, indices] <- values
  }

  return(S)
}
```

(iii) (5 marks) Using either of your methods from part (i) and part (ii), and making use of the knn function from the caret package (as in e.g. CW2):

(α) Randomly embed the data features into a subspace of dimension $r = 10$, and fit a K-Nearest Neighbours classifier which uses the embedded features $z_i = Sx_i$ to predict the label, performing a sweep over K to pick a good value of K.

```
library(class)
library(ggplot2)
library(lattice)
library(caret)
set.seed(50)

#Use dense embedding first
S_dense <- my_gauss_sketch(10,60)
dim(S_dense)
```

```
## [1] 10 60
```

```
#Use my_sse
S_sparse <- my_sse(10, 60)

#Ensure features is a matrix
X_train <- as.matrix(train_features)
X_test <- as.matrix(test_features)

#Compute embedded features
Z_dense_train <- X_train %*% t(S_dense)
Z_dense_test <- X_test %*% t(S_dense)

#Compute sparse features
Z_sparse_train <- X_train %*% t(S_sparse)
Z_sparse_test <- X_test %*% t(S_sparse)

#Convert labels into factors
train_labels <- factor(train_labels)
test_labels <- factor(test_labels)
```

```r
my_knn_sweep <- function(k_max, Z_train, Z_test){
  #Data frame to store reusults for each k
  results <- data.frame(k = integer(), Accuracy = numeric())

  #Loop for each k-value
  for (k in 1:k_max) {
    pred_knn <- knn(train = Z_train,
            test = Z_test,
            cl = train_labels,
            k = k)

    # Create a confusion matrix
    conf_Matrix <- confusionMatrix(pred_knn, test_labels)
    accuracy <- conf_Matrix$overall["Accuracy"]

    #Save the results
    results <- rbind(results, data.frame(k = k, Accuracy = as.numeric(accuracy)))

    #Determine k with the best accuracy
    best_index <- which.max(results$Accuracy)
    best_k <- results$k[best_index]
    best_accuracy <- results$Accuracy[best_index]
  }
  cat(sprintf("The best k is %d with an accuracy of %f.\n", best_k, best_accuracy))
  invisible(results)
}

k_max <- 50
my_knn_sweep(k_max, Z_dense_train, Z_dense_test)
```

```
## The best k is 1 with an accuracy of 0.865385.
```

```r
my_knn_sweep(k_max, Z_sparse_train, Z_sparse_test)
```

```
## The best k is 1 with an accuracy of 0.788462.
```

There was barely any difference of a good K-value between both methods. The best k in each method had an accuracy of more than 75%.

(β) Repeating this experiment 10 times, comment on the variability of the accuracy of your classifiers from one embedding to the next. Does it appear that the specific matrix S which we generate matters very much?

```r
set.seed(50)
E <- 10
k_max <- 50

#Create an empty array with dimensions 10 x 60 x 10
S_dense_array <- array(0, dim = c(10, 60, E))

for (e in 1:E) {
  #Use Dense Emebedding in this case
  S_dense_array[,,e] <- my_gauss_sketch(10, 60)
}

accuracy_results <- numeric(E)

#Looping through all possible embeddings
for (e in 1:E) {
  #Compute embedded features
  Z_dense_train <- X_train %*% t(S_dense_array[,,e])
  Z_dense_test <- X_test %*% t(S_dense_array[,,e])

  #Run k-NN sweep
  results_df <- my_knn_sweep(k_max, Z_dense_train, Z_dense_test)

  #Determine k with the best accuracy
  best_index <- which.max(results_df$Accuracy)
  accuracy_results[e] <- results_df$Accuracy[best_index]
}
```

```
## The best k is 1 with an accuracy of 0.865385.
## The best k is 1 with an accuracy of 0.884615.
## The best k is 1 with an accuracy of 0.807692.
## The best k is 1 with an accuracy of 0.846154.
## The best k is 3 with an accuracy of 0.826923.
## The best k is 1 with an accuracy of 0.807692.
## The best k is 2 with an accuracy of 0.846154.
## The best k is 2 with an accuracy of 0.826923.
## The best k is 38 with an accuracy of 0.769231.
## The best k is 3 with an accuracy of 0.769231.
```

```r
print(accuracy_results)
```

```
##  [1] 0.8653846 0.8846154 0.8076923 0.8461538 0.8269231 0.8076923 0.8461538
##  [8] 0.8269231 0.7692308 0.7692308
```

```r
#Print mean , sd, and range of our accuracy
mean_accuracy <- mean(accuracy_results)
std_accuracy <- sd(accuracy_results)
range_accuracy <- range(accuracy_results)

#Print Results
cat(sprintf("Mean Accuracy: %f\n", mean_accuracy))
```

```
## Mean Accuracy: 0.825000
```

```r
cat(sprintf("Standard Deviation: %f\n", std_accuracy))
```

```
## Standard Deviation: 0.037869
```

```r
cat(sprintf("Accuracy Range: %f to %f\n", range_accuracy[1], range_accuracy[2]))
```

```
## Accuracy Range: 0.769231 to 0.884615
```

The following are accuracy metrics for each embedding:

- Mean : 0.83
- Standard Deviation : 0.038
- Range : [0.77, 0.88]

It seems that on average the accuracy for K-NN using different independent embedding is >80%, which is highly desirable. The 0.039 standard deviation suggests that accuracy typically deviates 3.9 % away from the mean. The range is also consistent with the standard deviation, though there might be some that fall outside the range. Hence, generating specific matrix S doesn't really matter as there's not much variability in the accuracies.

(γ) (5 marks) For comparison, run K-Nearest Neigbours directly on the original features, again performing a sweep over K to pick a good value of K. Does this offer much of an improvement over the embedding-based classifiers?

```r
set.seed(50)
k_max <- 50

my_knn_sweep(k_max, X_train, X_test)
```

```
## The best k is 1 with an accuracy of 0.846154.
```

The accuracy produced here is 0.85. Compared to the mean of the accuracies from question β (0.83), it only offers a slight improvement over the embedding-based classifier.

(iv) Since the data set at hand is not too high-dimensional, it is feasible for us to compute the singular value decomposition of the original feature matrix using `svd`.

(α) Using the svd function to examine the structure of our original data set, and making particular reference to the singular values (which can be obtained via e.g. svd(features_matrix)$d), comment on why the accuracy of our embedding-based classifiers is not necessarily surprising.

```r
#Using original feature matrix
X <- sonar_data[,-ncol(sonar_data)]

#Using svd function
D <- svd(X)$d

print("Singular values:")
```

```
## [1] "Singular values:"
```

```r
print(D)
```

```
##  [1] 40.62628292 10.70937595  8.55873747  5.22294450  4.38161550  4.01841431
##  [7]  3.95531866  3.08788940  2.84903749  2.63780853  2.44678089  2.27954487
## [13]  2.07433464  1.86076429  1.78354939  1.67637642  1.61361118  1.43749096
## [19]  1.38499843  1.32931272  1.15226128  1.04256727  0.99363971  0.93280982
## [25]  0.91635239  0.88714849  0.75310264  0.71871814  0.69085263  0.66650772
## [31]  0.59237335  0.57639918  0.52676874  0.51654103  0.50343725  0.48647238
## [37]  0.44542881  0.43887913  0.42115088  0.38245289  0.37805320  0.36459419
## [43]  0.31574273  0.29070540  0.26395790  0.23188098  0.18242082  0.16882088
## [49]  0.16057580  0.13062871  0.10757103  0.09551877  0.08220268  0.07408432
## [55]  0.06664105  0.06214118  0.05337366  0.04943332  0.04415517  0.03468178
```

Singular values indicate the variance captured in each corresponding singular vector direction. Since many singular values are close to zero from our original feature matrix, it tells us that the rank of matrix is low and that much of the data structure can still be captured in a lower dimension. Reducing our dimension to $r = 10$ clearly still gives a roughly similar accuracy as it encompasses singular values from 2.63 to 40.63 which are large relative to the other 50 singular values.