Feature selection

Jan Izquierdo

2024-10-17

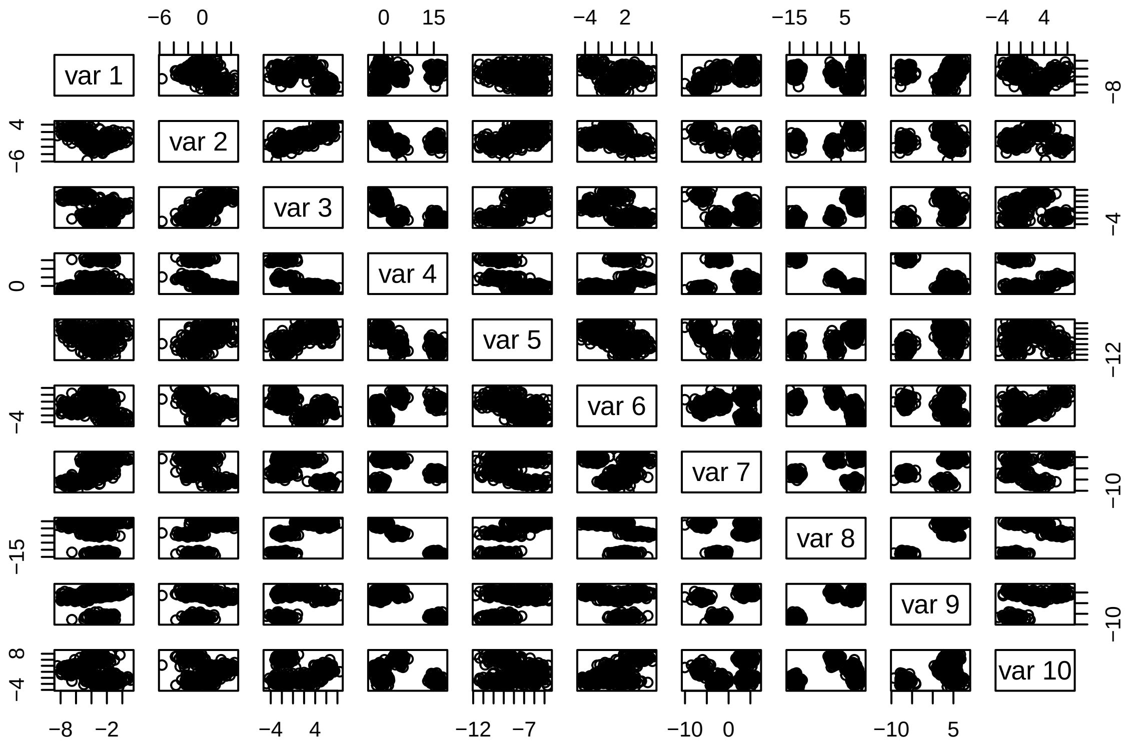
Task 1)

Create a dummy dataset of 400 items and 1000 features for KNN classification. Each item can belong to one of the two categories: A and B, each present at ~50% in your dataset. Only 10 features are interesting for KNN classification (i.e. it allows to classify the individuals in a KNN framework).

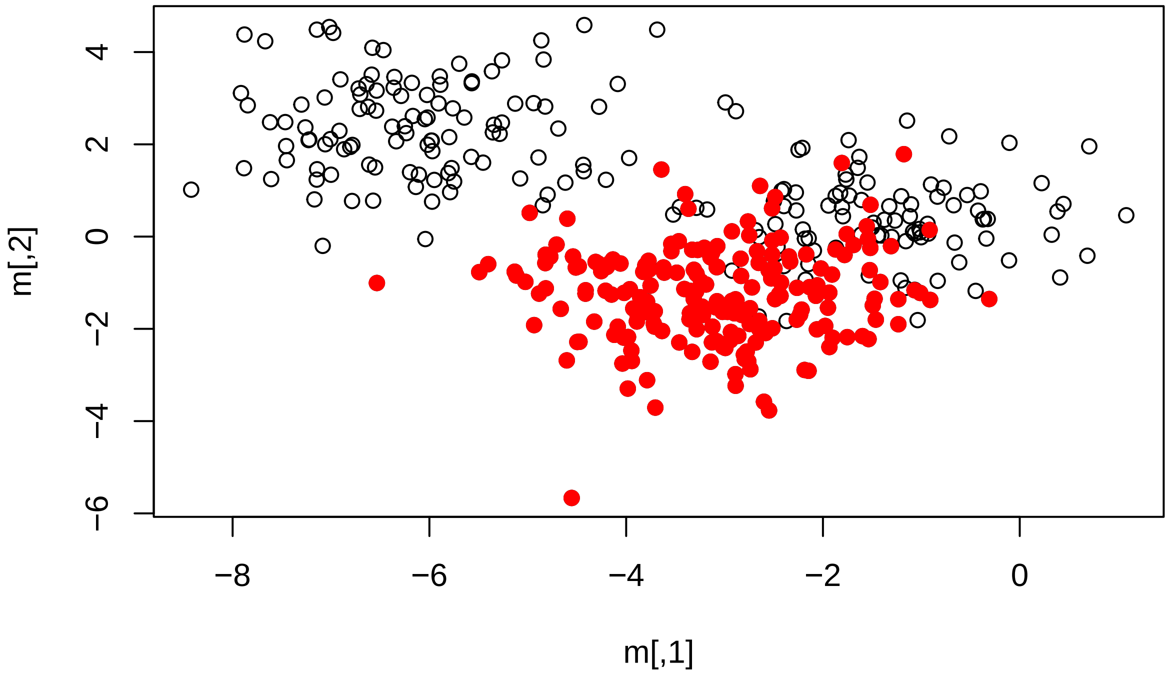
```r
library(MASS)
set.seed(2)
n_samples <- 400
# 400 items, what we want to predict
n_features <- 10
# simple to see if it is working or not , ind variables, variables we use
# to predict the independent values
n_patterns <- 2  #clusters per label
class_label <- as.factor(sample(c("A", "B"), n_samples, replace = T))
categories <- levels(class_label)  #wich options for discrete variable

m <- matrix(nrow = n_samples, ncol = n_features)  #To save generated data

for (cat in categories) {
    ids_cate <- which(class_label == cat)  #indexes for each category
    pattern <- sample(1:n_patterns, length(ids_cate), replace = T)
    for (pat in 1:n_patterns) {
        # count number of equal patterns to find important
        n_ids_cate_with_pattern <- sum(pattern == pat)
        mus <- rnorm(n_features, mean = 0, sd = 5)
        # the sd is large because that way the knn will be confused and
        # will focus on the ones we want
        X <- mvrnorm(n_ids_cate_with_pattern, mu = mus, Sigma = diag(n_features))
        # covariance matrix of 0s and 1s
        m[ids_cate[pattern == pat], ] <- X  #place new points in matrix
    }
}
pairs(m)
```



```
plot(m)
points(m[class_label == "A", 1], m[class_label == "A", 2], pch = 19, col = "red")
```



```
safe_m <- m  #save m in a backup variable just in case
```

All the remaining 990 features are useless for classification (contain noise). Make sure the noise has a bigger variance compared to the features that are interesting

```
m <- safe_m  #get m just in case it's been modified
random_features <- 1000 - n_features
```

```r
# take 990 -> 1000-10, we want to add 990 cols to the existing matrix
random_mean <- mean(m)
random_sd <- sd(m)  #you need a mean and sd to generate the data

# 990 times(one per column) create a vector of 400 features and add it to
# m as a column
for (n in 1:random_features) {
    random_feature <- rnorm(n_samples, random_mean, random_sd)
    m <- cbind(m, random_feature)
}
```

Divide your dataset in two. First will be use as training. Second as replication.

```r
library(class)
# get 70% of the data frame randomly as indices, then get the data using
# the indices and get the labels belonging to this indexes as well
train_index <- sample(1:n_samples, 0.7 * n_samples)  #70% data is for training
train_data <- m[train_index, ]  #get the data from the indices (each index is a row)
train_dLabels <- class_label[train_index]

# do the same with the remaining 30%
test_index <- setdiff(1:n_samples, train_index)  #30% data is for testing
test_data <- m[test_index, ]
test_dLabels <- class_label[test_index]
```

Task 2)

Run KNN with all the features. What happens?

```r
# use the knn predictor and get the accuracy of the model
pred_knn <- knn(train = train_data, test = test_data, cl = train_dLabels)
```

```
Predicted classes

 A  B
59 61

Actual classes

test_dLabels
 A  B
59 61

Accuracy of:  0.5833333
```

Task 3)

Create the mutation operation required for this implementation of ES. The mutation must pick one element at random from the chromosome and replace it by another element. BEWARE that the new element MUST NOT BE included in the set of already present elements in the chromosome!

```r
# Create function that selects random chromosome position, generates a
# random value unique in the chromosome and places in the randomly
# selected position
mutation <- function(data) {

    remaining_values <- setdiff(1:1000, data)  #get the unique value list

    rand_pos <- sample(1:length(data), 1)  #get the position

    unique_rand_num <- sample(remaining_values, 1)  #get the random number

    data[rand_pos] = unique_rand_num  #define the value in the chosen position

    return(data)
}

example_vec <- as.vector(seq(1, 1000))
example_vec <- sample(example_vec, 10)
example_vec_mut <- mutation(example_vec)
```

```
Pre mutation: 972 491 387 637 89 145 536 553 751 120


Post motation 972 491 387 637 89 145 150 553 751 120
```

Task 4)

Implement the phenotype function. Given the features selected in the chromosome, pick these in the training dataset.

```
# select columns from training data (columns are chromosomes)(we input
# indexes)
phenotype <- function(chr_index, traing_data) {
    selected_chr <- traing_data[, chr_index]

    return(selected_chr)
}

example_vec_Phe <- phenotype(example_vec, train_data)
head(example_vec_Phe)
```

```
random_feature random_feature random_feature random_feature random_feature
[1,] 6.2064642 -6.2684429 2.3550214 -3.307228 -8.2654817
[2,] -2.9077273 -1.1674287 -0.6612620 5.153358 -7.3490007
[3,] -2.6588069 -1.6774887 -1.0448262 -5.897920 -0.2836783
[4,] 1.7283502 -6.7136434 -0.8151014 7.182161 0.6758203
[5,] 0.4518524 -4.1566078 -8.1499693 -2.137181 1.1505718
[6,] -0.1252858 0.1429413 -1.2472105 4.696885 6.4350854
random_feature random_feature random_feature random_feature random_feature
[1,] 6.981867 6.703000 -2.481484 9.0522135 3.705444
[2,] 5.472085 -3.164119 4.390958 -4.5412585 2.264693
[3,] 4.899883 -2.584835 4.281086 0.1432859 1.832234
[4,] 6.262475 -8.015526 1.907107 -6.4624494 -8.058828
[5,] -2.725757 -1.131206 3.182324 -2.7358482 -2.697700
[6,] -3.053501 9.823839 6.768695 7.8524728 -1.894800
```

Task 5)

Implement the fitness function. Given the phenotype of a solution, run the KNN. The fitness is how good is the classification. The better the classification, the better the answer

```
# Implement the phenotype to filter the input data to only have the chr we
# are interested in(data from training) Then do a prediction using knn,
# and get the accuracy

fitness <- function(chr_index, training_data, test_data, train_class, test_class) {
    # Generate phentype
    phe_data <- phenotype(chr_index, training_data)

    # phenotype data is smaller, we reduce test as well
    selected_tst_data <- test_data[, chr_index]

    # run knn model with phenotype data
    pred_knn <- knn(train = phe_data, test = selected_tst_data, cl = train_class)

    return(mean(pred_knn == test_class))
}

acu <- fitness(example_vec, train_data, test_data, train_dLabels, test_dLabels)
```

```
The accuracy is of: 0.4583333
```

Task 6)

Implement the ES algorithm: At each iteration, pick one of the existing N (population size) solutions, each containing its own chromosome Allow this solution to create K new solutions by copying and mutating it. Add the new solutions to the pool of solutions. Now your population has N+K solutions. Rank the solutions given its fitness Pick the best N solutions. Remove the remaining ones. The iteration is finished At the end of the M iterations, report the solution with the best fitness over all the solutions Think about the hyperparameters you will need to do this implementation.

```
# Define generations, population size, etc
N <- 30  #Pop size
n_of <- 5  #Childs per generation
gen <- 50  #Number of generations
chr_len <- 20
total_chr_ind <- 1:1000  #what example_vec was previously

# Make function to create population
create_population <- function(N, chr_len, total_chr_ind) {
    population <- list()  #will be list of lists
    for (i in 1:N) {
        # create the population get chr_len number of chromosomes from the
        # total
        chr <- sample(total_chr_ind, chr_len)
        # save the chromosomes in a list in the total population list
        population[[i]] <- chr
    }
    return(population)
}

population <- create_population(N, chr_len, total_chr_ind)
length(population)
```

```
[1] 30
```

```
# Compute fitness for all populations
fitness_fromList <- function(N, pop, train_data, test_data, train_dLabels, test_dLabels) {
    total_fitness <- numeric(N)  #create vector
    for (i in 1:N) {
        pop <- population[[i]]  #compute finess for each case an place it in vector
        total_fitness[i] <- fitness(pop, train_data, test_data, train_dLabels,
            test_dLabels)
    }
    return(total_fitness)
}

total_fitness <- fitness_fromList(N, pop, train_data, test_data, train_dLabels,
    test_dLabels)
```

```
The average fitness for the inital population is 0.5211111
```

```
# evolution<-function(){} Do the whole evolutive process
for (i in 1:gen) {
    # select one of the N populations
    select_pop <- population[sample(1:N, 1)]

    # Mutate population to add offsprings
    newGen <- numeric(n_of)
    for (off in n_of) {
        newGen[off] <- mutation(select_pop)
    }

    # add new generation to population
    population <- c(population, newGen)
    # Compute fitness from newGen
    newGen_fit <- fitness_fromList(n_of, newGen, train_data, test_data, train_dLabels,
        test_dLabels)
    # add newGen fitness to existing
    total_fitness <- c(total_fitness, newGen_fit)

    # Pick best solutions

    # Rank by fitness
    ranked_indices_Fit <- order(total_fitness, decreasing = TRUE)

    # Pick best options
    best_pop <- population[ranked_indices_Fit[1:N]]
    best_fit <- total_fitness[ranked_indices_Fit[1:N]]
}

# Choose absolute best fitness and offspring
Test_fit <- max(best_fit)
Tbest_pop <- best_pop[[which.max(best_fit)]]
```

```
Best total fitness: 0.725


Best total population(its chr indexes): 670 516 19 683 665 159 630 870 511
869 886 437 565 364 8 628 743 88 491 475
```

Task 7)

Apply the ES-KNN algorithm to the dataset. Does it work?

```r
# Run KNN from best solution

# fit size of training data to same size as the best population
train_best <- phenotype(Tbest_pop, train_data)

# create reduced test as well
test_best <- test_data[, Tbest_pop]

# Run knn
pred_knn <- knn(train = train_best, test = test_best, cl = train_dLabels)
```

```
Accuracy for the best case is 0.725
```

```
The ES-KNN algorithm can be applied to the dataset.
```

Appendices

Extra information for performing the exercises

```r
library(MASS)

# Set seed for reproducibility
set.seed(42)

# Create a dummy regression dataset with 100 samples and 5 features
n_samples <- 100
n_features <- 2

# Generate random feature matrix (normally distributed data)
X <- mvrnorm(n_samples, mu = rep(0, n_features), Sigma = diag(n_features))

# Generate random target values with some noise
y <- X %*% runif(n_features) + rnorm(n_samples, sd = 0.1)

# Combine into a data frame
dummy_regression_data <- data.frame(X)
dummy_regression_data$Target <- y

# Print first few rows of the dataset
head(dummy_regression_data)

plot(dummy_regression_data$X1, dummy_regression_data$X2)

library(class)

# Split the dataset into training and testing sets (70% train, 30% test)
set.seed(42)
train_index <- createDataPartition(dummy_classification_data$Class, p = 0.7,
    list = FALSE)
train_data <- dummy_classification_data[train_index, ]
test_data <- dummy_classification_data[-train_index, ]
```