Homework 3

Antonio Padalino

2022-05-18

Contents

Homework 3

```r
# general purpose
library("tidyverse")

# plotting
library("ggplot2")
library("plotly")

# splitting and CV
library("rsample")
library("caret")

# support vectors
library("ISLR2")
library("e1071")
```

The `gene_expr.tsv` dataset comes from a study conducted on 79 patients with leukemia belonging to two groups (variable `y: -1/1`) representing the presence/absence of a chromosomal translocation. In order to make predictions on the subtype to which each patient belongs, the dataset features 2000 columns of gene expression data.

In the present examination SVMs will be used to model the relation between such genes expression and the 'target' subtypes.

1. Exploratory analysis

```r
df <- read.table("gene_expr.tsv", header = T)
```

Starting with a couple of observations:

- The column `sampleID` (unique ID of each patient) is useless for the analysis purpose and will be discarded.
- It may be useful to convert the `y` variable to a more 'classic' 0/1
  - after converting to 0/1, the `y` variable is made `as.factor()` in order to make this look like an actual classification problem by the `svm()` that will be called later

```r
df <- df %>%
    mutate(y = ifelse(y == -1, 0, 1))

df$y <- as.factor(df$y)

df <- df[, 2:2002]
```

2. Fitting a Support Vector Machine

With some data preparation done, SVM models are now fitted on the dataset.

In order to compare different models and get an actual measure of its performance, a train/test split is performed as usual (of course after setting a seed).

```r
set.seed(999)
```

```r
df_split <- initial_split(df, prop = 0.7)

df_train <- training(df_split)
df_test <- testing(df_split)
y_test <- df_test$y
x_test <- select(df_test, -y)
```

It might be interesting to try with different kernels and different hyperparameters to find the best SVM model possible. To do so, the `tune()` function comes at hand: it automatically performs 10 folds Cross Validation trying with a grid of provided values, and output the model performance on each try and the best parameters.

A. Linear Kernel

Let's try with a linear kernel first, passing to the `tune()` function a grid of values for `cost`.

```r
cost.grid <- seq(0.01, 1, 0.01)

tune.out <- tune(svm, y ~ ., data = df_train, kernel = "linear",
    ranges = list(cost = cost.grid))

tune.out$performances %>%
    summary()
```

```
##       cost            error        dispersion
##  Min.   :0.0100   Min.   :0.29   Min.   :0.2103
##  1st Qu.:0.2575   1st Qu.:0.29   1st Qu.:0.2103
##  Median :0.5050   Median :0.29   Median :0.2103
##  Mean   :0.5050   Mean   :0.29   Mean   :0.2103
##  3rd Qu.:0.7525   3rd Qu.:0.29   3rd Qu.:0.2103
##  Max.   :1.0000   Max.   :0.29   Max.   :0.2103
```

```
tune.out$best.parameters
```

cost
0.01

```
best.cost <- tune.out$best.parameters$cost

svm.fit1 <- svm(y ~ ., data = df_train, kernel = "linear", cost = best.cost)

acc1 <- mean(predict(svm.fit1, x_test) == y_test)

print(paste("Accuracy:", round(acc1, 4)))
```

```
## [1] "Accuracy: 0.75"
```

The accuracy is quite decent, but as one can see from the CV output above, the specified grid for cost values wasn't enough populated to notice any relevant values that might improve the model. Also, given the large number of columns and the overall runtime of the `tune()` function, a more fine-grained analysis (with a larger grid of values) is not possible.

B. Radial Kernel

One can also try with a radial kernel, with different values of `cost` and `gamma`:

```
cost.grid <- seq(0.01, 1, length = 10)
gamma.grid <- c(1250, 1000, 750)

tune.out <- tune(svm, y ~ ., data = df_train, kernel = "radial",
    ranges = list(cost = cost.grid, gamma = gamma.grid))

tune.out$performances %>%
    summary()
```

```
##       cost           gamma          error          dispersion
##  Min.   :0.010   Min.   : 750   Min.   :0.6967   Min.   :0.1856
##  1st Qu.:0.230   1st Qu.: 750   1st Qu.:0.6967   1st Qu.:0.1856
##  Median :0.505   Median :1000   Median :0.6967   Median :0.1856
##  Mean   :0.505   Mean   :1000   Mean   :0.6967   Mean   :0.1856
##  3rd Qu.:0.780   3rd Qu.:1250   3rd Qu.:0.6967   3rd Qu.:0.1856
##  Max.   :1.000   Max.   :1250   Max.   :0.6967   Max.   :0.1856
```

```r
tune.out$best.parameters
```

cost	gamma
0.01	1250

```r
best.cost <- tune.out$best.parameters$cost
best.gamma <- tune.out$best.parameters$gamma

svm.fit2 <- svm(y ~ ., data = df_train, kernel = "radial", cost = best.cost,
    gamma = best.gamma)

acc2 <- mean(predict(svm.fit2, x_test) == y_test)

print(paste("Accuracy:", round(acc2, 4)))
```

```
## [1] "Accuracy: 0.5417"
```

Again the error for different `cost` and `gamma` remains identical throughout the CV process. Choosing different hyperparameters once more seems not to have any effect on the final model. This time, the accuracy is even worse than before, around 0.54.

C. Polynomial Kernel

One can also try, lastly, with a polynomial kernel of different degrees.

```r
cost.grid <- seq(0.01, 1, length = 10)
gamma.grid <- c(1250, 1000, 750)
degree.grid <- c(2, 3)

tune.out <- tune(svm, y ~ ., data = df_train, kernel = "polynomial",
    ranges = list(cost = cost.grid, gamma = gamma.grid, degree = degree.grid))

tune.out$performances %>%
    summary()
```

```
##       cost            gamma           degree          error            dispersion
##  Min.   :0.010   Min.   : 750   Min.   :2.0   Min.   :0.5500   Min.   :0.1688
##  1st Qu.:0.230   1st Qu.: 750   1st Qu.:2.0   1st Qu.:0.5500   1st Qu.:0.1688
##  Median :0.505   Median :1000   Median :2.5   Median :0.5567   Median :0.1788
##  Mean   :0.505   Mean   :1000   Mean   :2.5   Mean   :0.5567   Mean   :0.1788
##  3rd Qu.:0.780   3rd Qu.:1250   3rd Qu.:3.0   3rd Qu.:0.5633   3rd Qu.:0.1887
##  Max.   :1.000   Max.   :1250   Max.   :3.0   Max.   :0.5633   Max.   :0.1887
```

```r
tune.out$best.parameters
```

	cost	gamma	degree
31	0.01	1250	3

```r
best.cost <- tune.out$best.parameters$cost
best.gamma <- tune.out$best.parameters$gamma
best.degree <- tune.out$best.parameters$degree

svm.fit3 <- svm(y ~ ., data = df_train, kernel = "polynomial",
    cost = best.cost, gamma = best.gamma, degree = best.degree)

acc3 <- mean(predict(svm.fit3, x_test) == y_test)

print(paste("Accuracy:", round(acc3, 4)))
```

```
## [1] "Accuracy: 0.7083"
```

The same pattern is found again, with no relevant changes in CV error with different hyperparameters.

This may be due to a number of reasons, the first one being the large number of features in the given dataset. For such motive, one may think to remove from the dataset the columns referring to the less variable factors (in other words, discarding gene expression data from genes that are the less variable).

To do so programmatically, in the next paragraph some features will be removed according to their standard deviation.

Once more, notice that another reason causing the pattern above may also be the choice of grids for each hyperparameter. However, using a larger, more fine-grained grid would have been much more complex computationally speaking.

3. SVM on reduced dataset

A popular approach on such kinds of analysis is to reduce the number of columns (gene expression data) considered in the model, since most of them do not vary much among different subjects.

For such reason, the dataset is now reduced to only contain features within the top 5% standard deviation.

```r
quantile <- sapply(select(df, -y), sd) %>%
    sort() %>%
    quantile(0.95)

reduced.df <- select(df, -y) %>%
    select_if(sapply(select(df, -y), sd) >= quantile)
reduced.df <- cbind(reduced.df, y = df$y)
```

```r
df_split <- initial_split(reduced.df, prop = 0.7)

df_train.red <- training(df_split)
df_test.red <- testing(df_split)
y_test.red <- df_test.red$y
x_test.red <- select(df_test.red, -y)
```

After selecting only the top 5% highest-sd columns, one can repeat the same procedure seen above, to fit different SVM models.

The workflow for the following part will be:

→ using `tune()` on a grid of values to find different hyperparameters (using training data)

→ fitting a SVM with such hyperparameters (always with training data)

→ computing its accuracy on test dataset

A. Linear Kernel - red. dataset

Starting with a linear kernel:

```
cost.grid <- seq(0.001, 0.2, 0.005)  # less columns -> we can afford a bigger grid this time

tune.out <- tune(svm, y ~ ., data = df_train.red, kernel = "linear",
    ranges = list(cost = cost.grid))

tune.out$performances %>%
    summary()
```

```
##       cost              error          dispersion
##  Min.   :0.00100   Min.   :0.2367   Min.   :0.1688
##  1st Qu.:0.04975   1st Qu.:0.2533   1st Qu.:0.2021
##  Median :0.09850   Median :0.2567   Median :0.2079
##  Mean   :0.09850   Mean   :0.2592   Mean   :0.2004
##  3rd Qu.:0.14725   3rd Qu.:0.2567   3rd Qu.:0.2079
##  Max.   :0.19600   Max.   :0.5000   Max.   :0.2079
```

```
tune.out$best.parameters
```

	cost
4	0.016

```
best.cost <- tune.out$best.parameters$cost

svm.fit1r <- svm(y ~ ., data = df_train.red, kernel = "linear",
    cost = best.cost)

acc1r <- mean(predict(svm.fit1r, x_test.red) == y_test.red)

print(paste("Accuracy:", round(acc1r, 4)))
```

```
## [1] "Accuracy: 0.8333"
```

B. Radial Kernel - red. dataset

Now for a radial kernel:

```r
cost.grid <- seq(0.001, 1, 0.01)
gamma.grid <- c(75, 100, 125)

tune.out <- tune(svm, y ~ ., data = df_train.red, kernel = "radial",
    ranges = list(cost = cost.grid, gamma = gamma.grid))

tune.out$performances %>%
    summary()
```

```
##       cost             gamma         error          dispersion
##  Min.   :0.0010   Min.   : 75   Min.   :0.5433   Min.   :0.1483
##  1st Qu.:0.2485   1st Qu.: 75   1st Qu.:0.5433   1st Qu.:0.1483
##  Median :0.4960   Median :100   Median :0.5433   Median :0.1483
##  Mean   :0.4960   Mean   :100   Mean   :0.5433   Mean   :0.1483
##  3rd Qu.:0.7435   3rd Qu.:125   3rd Qu.:0.5433   3rd Qu.:0.1483
##  Max.   :0.9910   Max.   :125   Max.   :0.5433   Max.   :0.1483
```

```r
tune.out$best.parameters
```

cost	gamma
0.001	75

```r
best.cost <- tune.out$best.parameters$cost
best.gamma <- tune.out$best.parameters$gamma

svm.fit2r <- svm(y ~ ., data = df_train.red, kernel = "radial",
    cost = best.cost, gamma = best.gamma)

acc2r <- mean(predict(svm.fit2r, x_test.red) == y_test.red)

print(paste("Accuracy:", round(acc2r, 4)))
```

```
## [1] "Accuracy: 0.5417"
```

C. Polynomial Kernel - red. dataset

Now for a polynomial kernel:

```r
cost.grid <- seq(0.001, 1, 0.01)
gamma.grid <- seq(50, 150, 10)
degree.grid <- c(2, 3)

tune.out <- tune(svm, y ~ ., data = df_train.red, kernel = "polynomial",
    ranges = list(cost = cost.grid, gamma = gamma.grid, degree = degree.grid))

tune.out$performances %>%
    summary()
```

```
##        cost            gamma          degree          error           dispersion
##  Min.   :0.0010   Min.   : 50   Min.   :2.0   Min.   :0.3200   Min.   :0.1793
##  1st Qu.:0.2485   1st Qu.: 70   1st Qu.:2.0   1st Qu.:0.3200   1st Qu.:0.1793
##  Median :0.4960   Median :100   Median :2.5   Median :0.4067   Median :0.1798
##  Mean   :0.4960   Mean   :100   Mean   :2.5   Mean   :0.4067   Mean   :0.1798
##  3rd Qu.:0.7435   3rd Qu.:130   3rd Qu.:3.0   3rd Qu.:0.4933   3rd Qu.:0.1804
##  Max.   :0.9910   Max.   :150   Max.   :3.0   Max.   :0.4933   Max.   :0.1804
```

```
tune.out$best.parameters
```

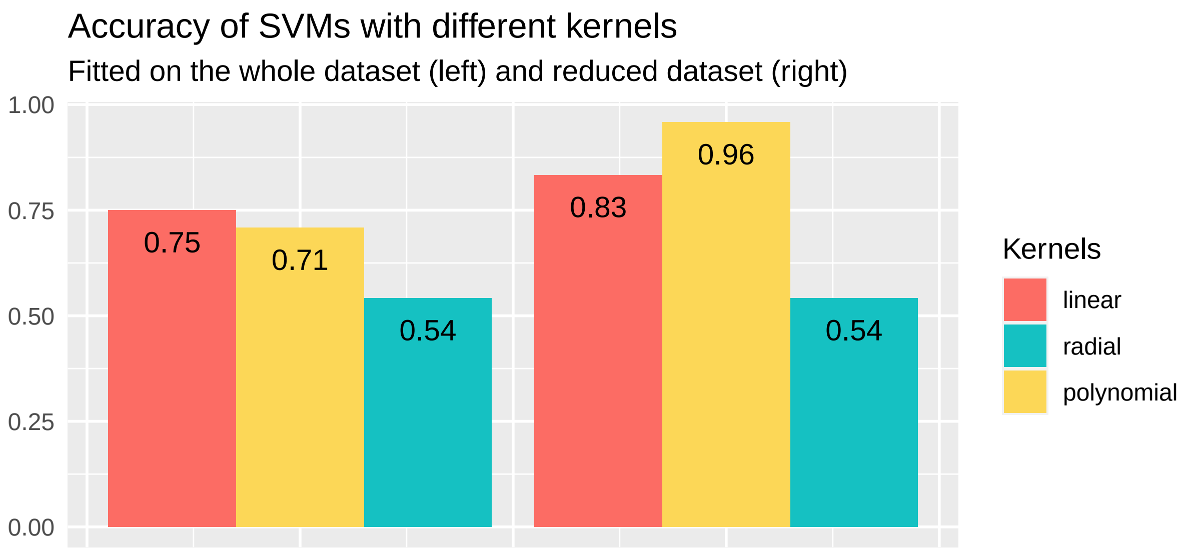
	cost	gamma	degree
1101	0.001	50	3

```
best.cost <- tune.out$best.parameters$cost
best.gamma <- tune.out$best.parameters$gamma
best.degree <- tune.out$best.parameters$degree

svm.fit3r <- svm(y ~ ., data = df_train.red, kernel = "polynomial",
    cost = best.cost, gamma = best.gamma, degree = best.degree)

acc3r <- mean(predict(svm.fit3r, x_test) == y_test)

print(paste("Accuracy:", round(acc3r, 4)))
```

```
## [1] "Accuracy: 0.9583"
```

4. General considerations

```
to_plot <- data.frame(error = c(acc1, acc2, acc3, acc1r, acc2r,
    acc3r), reduced = c(rep(0, 3), rep(1, 3)), kernel = c(rep(c("linear",
    "radial", "polynomial"), 2)))

ggplot(to_plot, aes(fill = kernel, y = error, x = reduced)) +
    geom_bar(position = "dodge", stat = "identity") + labs(title = "Accuracy of SVMs with different kerr
    x = "", y = "", subtitle = "Fitted on the whole dataset (left) and reduced dataset (right)") +
    scale_fill_manual("Kernels", values = c(linear = "#FC6C64",
        radial = "#15C1C2", polynomial = "#FCD757")) + geom_text(aes(label = round(error,
    2)), position = position_dodge(width = 0.9), vjust = 2) +
    theme(axis.text.x = element_blank(), axis.ticks = element_blank(),
        aspect.ratio = 0.5)
```



Generally, the accuracies of SVMs fitted on the reduced dataset containing only the most variable features are higher than those of SVMs fitted on the whole dataset. The reduced dataset has less noise - so to say - and also allows a more in depth research of the best hyperparameters (since less features $\rightarrow$ less time to tune).

One can say that, overall:

- a polynomial kernel SVM seems like the best choice overall (with an accuracy of 0.96 in this analysis)
- a radial kernel SVM is probably the less recommended model in this scenario, as it scores pretty low (accuracy of 0.54) on both datasets