Homework.3

Eleonora Giuliani 247161

2024-05-13

This study analyzes a gene expression dataset consisting of 79 patients diagnosed with leukimia. The dataset comprises expression data for 2,000 genes, alongside patient labels indicating two distinct leukemia subgroups: patients with a chromosomal translocation (labelled as "1") and patients with cytogenetically normal leukemia (labelled as "-1"). The objective of this analysis is to develop a predictive model using support vector machines (SVMs) to classify patients into their respective leukemia subgroups based on their gene expression profiles.

Before properly starting the analysis, we shall briefly examine the present dataset, beginning verifying the correctness of the opening.

```
[1]   79 2002
```

It is also important to verify the presence of NA values. In this case they are not present. Lastly, we check that the last column contains information about the translocation status of each patient.

```
[1] "y"
```

```
[1]  1 -1
```

We proceeded, with splitting the dataset using the 'sample' function, allocating 70% to the training set and 30% to the testing set.To ensure reproducibility, we set the seed to a specific value, in particular 1.

```
set.seed(1)
train_proportion <- 0.7  # 70% for training, 30% for testing
# Calculate the number of samples for training
n_train <- round(train_proportion * nrow(dataf))
# Generate random indices for selecting samples for training
```

```
train_indices <- sample(1:nrow(dataf), n_train, replace = FALSE)

xtrain <- dataf[train_indices, ]
xtest <- dataf[-train_indices, ]
ytrain <- dataf$y[train_indices]
ytest <- dataf$y[-train_indices]
```

Now, we create a dataframe containing only the training data. It is also necessary to convert ytrain into a factor.

```
dat <- data.frame(x = xtrain, y = as.factor(ytrain))
```

At this point, it is possible to evaluate different models using Support Vector Machine. To utilize SVM, it is necessary to load the 'e1071' library.

```
library("e1071")
```

We begin by setting the kernel as linear. To identify the optimal model, we conduct cross-validation using the 'tune()' function. The 'svm' function is specified as the model to be tuned. 'y' is the response variable and all the other columns in the dataframe are utilized as predictors.

```
set.seed(1)
cost_range <- c(0.001, 0.01, 0.1, 1, 5, 10, 100)
tune.out <- tune(svm, y ~ ., data = dat, kernel = "linear", ranges = list(cost = cost_range)
```

The cross-validation process allows us to identify the optimal value for the cost parameter. In this case the optimal cost is 0.01 with an error of 0.1633333

```
bestmod <- tune.out$best.model
summary(bestmod)
```

```


Call:
best.tune(METHOD = svm, train.x = y ~ ., data = dat, ranges = list(cost = cost_range),
    kernel = "linear")


Parameters:
   SVM-Type:  C-classification
```

```
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  49

 ( 28 21 )


Number of Classes:  2

Levels:
 -1 1
```

Once we have determined the best-tuned SVM model with the optimal cost parameter, we can proceed to evaluate its performance computing the confusion matrix and the accuracy of the model.

```
pred.te <- predict(bestmod, newdata = dat)
table(pred.te, dat$y)
```

```

pred.te -1  1
     -1 33  0
     1   0 22
```

```
mean(pred.te == dat$y)
```

```
[1] 1
```

The accuracy of the model is 100%, indicating that all predictions made by the model on the training dataset were correct. However, it is important to evaluate the model's performance on the test dataset.

Indeed, we apply the model on the dataframe containing the test and, again, it performs perfectly.

```
dat.te <- data.frame(x = xtest, y = as.factor(ytest))
svm_linear <- svm(y ~ ., data = dat.te, kernel = "linear", cost = 0.01)

pred.te <- predict(svm_linear, newdata = dat.te)
table(pred.te, dat.te$y)
```

```
pred.te -1  1
     -1  9  0
     1   0 15
```

```
mean(pred.te == dat.te$y)
```

```
[1] 1
```

The next step is to evaluate the SVM model with the radial kernel on the training dataset. In this evaluation, cross-validation will help identify the optimal values not only for the cost parameter but also for the gamma parameter.

```
set.seed(1)
cost_range <- c(0.1, 1, 10, 100, 1000)
gamma_range <- c(0.5, 1, 2, 3, 4)
tune.out <- tune(svm, y ~ ., data = dat, kernel = "radial", ranges = list(cost = cost_range,
    gamma = gamma_range))
```

The optimal parameters are cost = 0.1 and gamma = 0.5 with an error of 0.4.

The following code computes predictions using the best-tuned SVM model on the train data. Then, we evaluate the performance of the model.

```
ypred <- predict(tune.out$best.model, newdata = dat)
table(true = dat$y, pred = ypred)
```

```
    pred
true -1  1
  -1 33  0
  1  22  0
```

```
mean(dat$y == ypred)
```

```
[1] 0.6
```

In this case, the model exhibits low accuracy and it misclassifies approximately 40% of the samples. To conduct further investigation, we evaluate the model's performance on the test data as well.

```r
svm_radial <- svm(y ~ ., data = dat.te, kernel = "radial", cost = 0.1, gamma = 0.5)
```

In both scenarios, the models achieve an accuracy of around 60%, indicating that approximately 60% of the predictions made by the models are correct. However, the first model fails to recognize any negative instances(class -1), while the second model fails to recognize any positive instances (class 1).

```r
ypred <- predict(svm_radial, newdata = dat.te)
table(true = dat.te$y, pred = ypred)
```

```
    pred
true -1  1
  -1  0  9
  1   0 15
```

```r
mean(dat.te$y == ypred)
```

```
[1] 0.625
```

Lastly, we decide to analyze the SVM model with the kernel set to 'polynomial'. In this scenario, cross-validation is utilized to identify the optimal values for both the cost and degree parameters.

```r
set.seed(1)
cost_range <- c(0.1, 1, 10, 100, 1000)
degree_range <- c(1, 2, 3, 4, 5)

tune.out.pol <- tune(svm, y ~ ., data = dat, kernel = "polynomial", ranges = list(cost = cos
    degree = degree_range))
```

The optimal parameters are cost = 10 and degree = 1 with an error of 0.1633333.

Computing the accuracy and the confusion matrix, it appears that the model perform perfectly, achieving an accuracy of 1.

```r
ypred <- predict(tune.out.pol$best.model, newdata = dat)
table(true = dat$y, pred = ypred)
```

```
    pred
true -1  1
  -1 33  0
  1   0 22
```

```r
mean(dat$y == ypred)
```

```
[1] 1
```

This trend is confirmed when evaluating the model on the test data.

```r
svm_polynomial <- svm(y ~ ., data = dat.te, kernel = "polynomial", cost = 10, degree = 1)

ypred <- predict(svm_polynomial, newdata = dat.te)
table(dat.te$y, ypred)
```

```
    ypred
     -1  1
  -1  9  0
  1   0 15
```

```r
mean(dat.te$y == ypred)
```

```
[1] 1
```

A popular approach in gene expression analysis is to keep only the most variable genes for downstream analysis. Since most of the 2K genes have low expression or do not vary much across the experiments, this step usually minimizes the contribution of noise. Indeed, we select then only genes whose standard deviation is among the top 5% and we repeat the analyses performed previously on the filtered data set.

We obtain a dataset with the following dimension:

```
[1]  79 102
```

As before, it is necessary to transform y in a factor.

```
fil_df$y <- as.factor(fil_df$y)
```

We proceed applying the SVM, initially with the linear kernel.

```
set.seed(1)
cost_range <- c(0.001, 0.01, 0.1, 1, 5, 10, 100)
tune.out_linear <- tune(svm, y ~ ., data = fil_df, kernel = "linear", ranges = list(cost = co
```

As for the entire dataset, the optimal cost parameter is 0.01. However, this time, the model does not perform perfectly. There are approximately 10% of incorrect predictions.

```
# performance
pred <- predict(tune.out_linear$best.model, newdata = fil_df)
table(pred, fil_df$y)
```

```
pred -1  1
  -1 40  6
  1   2 31
```

```
mean(pred == fil_df$y)
```

```
[1] 0.8987342
```

Then, we apply the SVM model with the radial kernel.

```
set.seed(1)
cost_range <- c(0.1, 1, 10, 100, 1000)
gamma_range <- c(0.5, 1, 2, 3, 4)
tune.out_radial <- tune(svm, y ~ ., data = fil_df, kernel = "radial", ranges = list(cost = co
    gamma = gamma_range))
```

Again, the optimal parameters are cost=0.1 and gamma=0.5. However, the model accuracy is lower when using the filtered dataset.

```
# performance
ypred <- predict(tune.out_radial$best.model, newdata = fil_df)
table(true = fil_df$y, pred = ypred)
```

```
    pred
true -1  1
  -1 42  0
  1  37  0
```

```
mean(fil_df$y == ypred)
```

```
[1] 0.5316456
```

Lastly, we evaluate the polynomial kernel.

```
set.seed(1)
cost_range <- c(0.1, 1, 10, 100, 1000)
degree_range <- c(1, 2, 3, 4, 5)

tune.out.pol <- tune(svm, y ~ ., data = fil_df, kernel = "polynomial", ranges = list(cost = 
    degree = degree_range))
```

In this case, the optimal parameters are different from those for the full dataset (cost = 10 and degree = 3), but the model still performs perfectly.

```
# performance
ypred <- predict(tune.out.pol$best.model, newdata = fil_df)
table(true = fil_df$y, pred = ypred)
```

```
    pred
true -1  1
  -1 42  0
  1   0 37
```

```
mean(fil_df$y == ypred)
```

```
[1] 1
```

“‘

Conclusion

Overall, while the polynomial SVM model consistently performed well and achieved perfect accuracy in both dataset scenarios, the linear SVM model showed a slight decrease in accuracy when applied to the filtered dataset, indicating some sensitivity to feature selection. However, the radial SVM model exhibited lower accuracy despite having optimal parameters. Indeed, the radial SVM model struggled to classify the samples correctly, suggesting potential limitations in capturing the underlying patterns in the data.