Statistical Learning, Homework #3

Annalisa Xamin

Introduction

In the following analysis we will be working on a gene expression data set of 79 patients with leukemia belonging to two groups: patients with a chromosomal translocation (marked as 1) and patients cytogenetically normal (marked as -1). The data in `gene_expr.tsv` file, contains the expression for 2,000 genes and an additional column with the subtype.

To begin, we load the data

```r
gene_df <- read.csv("./gene_expr.tsv", sep = "")
dim(gene_df)
```

```
## [1]   79 2002
```

We have 2002 columns: 2000 of them contains the gene expression, while the other 2 are the sampleID column, which identify the patient, and the response variable (y) column, which divides the patients into two groups.
We are dealing with high-dimensional data, because the number of features (2000) exceed the number of observations (79).

We checked that in the data set there are no NAs. Moreover, from the summary we can see that the classes are quite balanced: we have 42 observations of patients cytogenetically normal and 37 of patients with a chromosomal translocation.

Part 1: Whole dataset

Data split into training and test sets

We divide the data in training and test sets, assigning 70% of the data set to the training data.

```r
set.seed(1)
sample <- initial_split(gene_df, prop = 7/10)
train_data <- training(sample)
test_data <- testing(sample)
train_splits <- vfold_cv(train_data, v = 10)
```

The `train_splits` will be used in the tuning of the hyperparameters using a 10-fold cross validation.

Since we will later use the `tune_grid` function to tune the parameters, we will need a “recipe” to pre-process the data. In particular, we specify that the column `sampleID` shouldn’t be used as predictor, but just as identifier for the observations. Moreover, the response variable `y` is converted to factor.

```r
preprocessor <- recipe(y ~ ., data = train_data) %>%
    add_role(sampleID, new_role = "ID") %>%
    step_bin2factor(y, levels = c("1", "-1"))
preprocessor <- preprocessor %>%
    prep()
```

Tuning of the hyperparameters

We will classify each individual using Support Vector Machines with 3 different kernels: linear, radial and polynomial. For each kernel, we want to find the best tuning parameters: the `cost` is shared by all three kernels, the radial adds the `gamma` and the polynomial the `degree`.

A grid of dimensions $v \times p$ will be produced for each model, where v represents the total number of feasible parameter combinations and p denotes the number of parameters associated with the model. Upon generating the grid, the function `tune_grid` is employed to execute a 10-fold cross-validation process for each combination of tuning parameters, utilizing the previously generated splits. The function `tune_grid` will generate a tibble that includes the tuning outcomes, which will be stored in the column labeled `.metrics`. Given that we are dealing with a binary classification problem, the focus of this particular study is solely on the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve. The objective is to identify the optimal model that can effectively distinguish the test observations.

During the tuning step, each model will have the same range of values of cost and the same training splits.

For each kernel, the optimal parameters are reported in a table.

Linear Kernel

```r
linear_svm_tune <- svm_linear(cost = tune()) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

svm_linear_grid <- grid_regular(cost(c(-10, 5)), levels = 5)

res_linear <- tune_grid(linear_svm_tune, preprocessor, resamples = train_splits,
    grid = svm_linear_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_linear))
```

cost	.config
0.013139	Preprocessor1_Model2

Polynomial kernel

```r
poly_svm_tune <- svm_poly(cost = tune(), degree = tune()) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

par <- parameters(cost(c(-10, 5)), degree(c(2, 5)))

poly_grid <- grid_regular(par, levels = c(cost = 5, degree = 4))
```

```r
res_poly <- tune_grid(poly_svm_tune, preprocessor, resamples = train_splits,
    grid = poly_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_poly))
```

cost	degree	.config
0.013139	3	Preprocessor1_Model07

Radial kernel

We can write `gamma` as $\gamma = \frac{1}{2\sigma^2}$. So, tuning `gamma` is equivalent of tuning `sigma`. To find the optimal value of `sigma`, we should use a large range of values, but in our case, to reduce the time needed for the computation, we use a smaller arbitrary range, between 0.0001 and 0.00045.

```r
radial_svm_tune <- svm_rbf(cost = tune(), rbf_sigma = tune()) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

par <- parameters(cost(c(-10, 5)), rbf_sigma(c(-4, -3.35)))

radial_grid <- grid_regular(par, levels = c(cost = 5, rbf_sigma = 5))

res_radial <- tune_grid(radial_svm_tune, preprocessor, resamples = train_splits,
    grid = radial_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_radial))
```

cost	rbf_sigma	.config
32	1e-04	Preprocessor1_Model05

The optimal value of `sigma` is the smallest as possible between the ones proposed, indicating a large value of `gamma`. When γ takes on a large value, the decision boundaries are primarily influenced by individual training observations.

Build the models

We pre-process the data as follow:

```r
train_data_juiced <- preprocessor %>%
    juice()
test_data_baked <- preprocessor %>%
    bake(test_data)
```

Then, we select the best parameters for each kernel and use them to build the models.

```r
linear_best_par <- select_best(res_linear)
poly_best_par <- select_best(res_poly)
radial_best_par <- select_best(res_radial)
```

```r
linear_svm <- svm_linear(cost = linear_best_par$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

poly_svm <- svm_poly(degree = poly_best_par$degree, cost = poly_best_par$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

radial_svm <- svm_rbf(rbf_sigma = radial_best_par$rbf_sigma,
    cost = radial_best_par$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")
```

Results

For each kernel, now we fit the SVM and retrieve confusion matrix, accuracy, AUC and ROC curve.

Linear SVM

```r
linear_svm_fit <- linear_svm %>%
    fit(formula = y ~ ., data = train_data_juiced)
```

```
##  Setting default kernel parameters
```

```r
# confusion matrix
linear_svm_fit %>%
    augment(new_data = test_data_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##             Truth
## Prediction 1 -1
##          1  7  0
##         -1 8  9
```

```r
lin_augmented <- linear_svm_fit %>%
    augment(new_data = test_data_baked)

# accuracy
accuracy_linear <- lin_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_linear <- accuracy_linear$.estimate  # 0.6666667

# AUC
linear_svm_auc <- lin_augmented %>%
    roc_auc(y, .pred_1)
linear_svm_auc <- linear_svm_auc$.estimate  # 0.9037037

# ROC curve
linaer_roc <- lin_augmented %>%
    roc_curve(y, .pred_1)
```

Polynomial SVM

```r
poly_svm_fit <- poly_svm %>%
    fit(formula = y ~ ., data = train_data_juiced)

# confusion matrix
poly_svm_fit %>%
    augment(new_data = test_data_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##            Truth
## Prediction 1 -1
##         1  9  3
##         -1 6  6
```

```r
poly_augmented <- poly_svm_fit %>%
    augment(new_data = test_data_baked)

# accuracy
accuracy_poly <- poly_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_poly <- accuracy_poly$.estimate  # 0.625

# AUC
poly_svm_auc <- poly_augmented %>%
    roc_auc(y, .pred_1)
poly_svm_auc <- poly_svm_auc$.estimate  # 0.748148

# ROC curve
poly_roc <- poly_augmented %>%
    roc_curve(y, .pred_1)
```

Radial SVM

```r
radial_svm_fit <- radial_svm %>%
    fit(formula = y ~ ., data = train_data_juiced)

# confusion matrix
radial_svm_fit %>%
    augment(new_data = test_data_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##            Truth
## Prediction 1 -1
##         1  8  0
##         -1 7  9
```

```r
radial_augmented <- radial_svm_fit %>%
    augment(new_data = test_data_baked)
```

```r
# accuracy
accuracy_radial <- radial_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_radial <- accuracy_radial$.estimate  # 0.7083333

# AUC
radial_svm_auc <- radial_augmented %>%
    roc_auc(y, .pred_1)
radial_svm_auc <- radial_svm_auc$.estimate  # 0.8814815

# ROC curve
radial_roc <- radial_augmented %>%
    roc_curve(y, .pred_1)
```

The radial kernel is the SVM with higher accuracy. However, it is the linear kernel the one with better AUC value.

Part 2: Reducing variables by removing noise

In this part of the analysis, we repeat the analyses performed in the previous part, filtering the data set and keeping only the most variable genes. In particular, we select only genes whose standard deviation is among the top 5%.

```r
ncols <- dim(gene_df)[2]
genes_sd <- gene_df %>%
    summarise(across(c(3:ncols - 1), .fns = sd))
genes_sd <- genes_sd %>%
    tidyr::pivot_longer(everything(), names_to = "Gene", values_to = "sd") %>%
    filter(sd >= quantile(sd, c(0.95, 1))[1])

gene_df_filtered <- gene_df %>%
    select(sampleID, contains(genes_sd$Gene), y)


# split into test and training
set.seed(1)
genes_filt_splits <- initial_split(gene_df_filtered, prop = 7/10)
train_filt_data <- training(genes_filt_splits)
test_filt_data <- testing(genes_filt_splits)
train_filt_splits <- vfold_cv(train_filt_data, v = 10)

# recipe needed to build the models
preprocessor_filt <- recipe(y ~ ., data = train_filt_data) %>%
    add_role(sampleID, new_role = "ID") %>%
    step_bin2factor(y, levels = c("1", "-1"))
preprocessor_filt <- preprocessor_filt %>%
    prep()
```

Tuning of hyperparameters

Linear kernel

```
res_linear_filt <- tune_grid(linear_svm_tune, preprocessor_filt,
    resamples = train_filt_splits, grid = svm_linear_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_linear_filt))
```

cost	.config
0.1767767	Preprocessor1_Model3

Polynomial kernel

```
res_poly_filt <- tune_grid(poly_svm_tune, preprocessor_filt,
    resamples = train_filt_splits, grid = poly_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_poly_filt))
```

cost	degree	.config
0.0009766	5	Preprocessor1_Model16

Radial kernel

```
res_radial_filt <- tune_grid(radial_svm_tune, preprocessor_filt,
    resamples = train_filt_splits, grid = radial_grid, metrics = metric_set(roc_auc))
knitr::kable(select_best(res_radial_filt))
```

cost	rbf_sigma	.config
32	0.0003073	Preprocessor1_Model20

Build the models

```
train_data_filt_juiced <- preprocessor_filt %>%
    juice()
test_data_filt_baked <- preprocessor_filt %>%
    bake(test_data)

# extract best parameters
linear_best_par_filt <- select_best(res_linear_filt)
poly_best_par_filt <- select_best(res_poly_filt)
radial_best_par_filt <- select_best(res_radial_filt)

# build models
```

```
linear_svm_filt <- svm_linear(cost = linear_best_par_filt$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

poly_svm_filt <- svm_poly(degree = poly_best_par_filt$degree,
    cost = poly_best_par_filt$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")

radial_svm_filt <- svm_rbf(rbf_sigma = radial_best_par_filt$rbf_sigma,
    cost = radial_best_par_filt$cost) %>%
    set_mode("classification") %>%
    set_engine("kernlab")
```

Results

For each kernel, now we fit the SVM and retrieve confusion matrix, accuracy, AUC and ROC curve.

Linear SVM

```
linear_svm_filt_fit <- linear_svm_filt %>%
    fit(formula = y ~ ., data = train_data_filt_juiced)
```

```
##  Setting default kernel parameters
```

```
# confusion matrix
linear_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##           Truth
## Prediction  1 -1
##         1  10  1
##         -1  5  8
```

```
lin_filt_augmented <- linear_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked)

# accuracy
accuracy_linear_filt <- lin_filt_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_linear_filt <- accuracy_linear_filt$.estimate  # 0.75

# AUC
linear_svm_auc_filt <- lin_filt_augmented %>%
    roc_auc(y, .pred_1)
linear_svm_auc_filt <- linear_svm_auc_filt$.estimate  # 0.8296296

# ROC curve
```

```
linaer_roc_filt <- lin_filt_augmented %>%
    roc_curve(y, .pred_1)
```

Polynomial SVM

```
poly_svm_filt_fit <- poly_svm_filt %>%
    fit(formula = y ~ ., data = train_data_filt_juiced)

# confusion matrix
poly_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##             Truth
## Prediction   1 -1
##          1  15  7
##         -1   0  2
```

```
poly_filt_augmented <- poly_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked)

# accuracy
accuracy_poly_filt <- poly_filt_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_poly_filt <- accuracy_poly_filt$.estimate  # 0.7083333

# AUC
poly_svm_auc_filt <- poly_filt_augmented %>%
    roc_auc(y, .pred_1)
poly_svm_auc_filt <- poly_svm_auc_filt$.estimate  # 0.9111111

# ROC curve
poly_roc_filt <- poly_filt_augmented %>%
    roc_curve(y, .pred_1)
```

Radial SVM

```
radial_svm_filt_fit <- radial_svm_filt %>%
    fit(formula = y ~ ., data = train_data_filt_juiced)

# confusion matrix
radial_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked, estimate = ".pred_class") %>%
    conf_mat(truth = y, estimate = .pred_class)
```

```
##             Truth
## Prediction   1 -1
##          1  11  0
##         -1   4  9
```

```r
radial_filt_augmented <- radial_svm_filt_fit %>%
    augment(new_data = test_data_filt_baked)

# accuracy
accuracy_radial_filt <- radial_filt_augmented %>%
    accuracy(truth = y, estimate = ".pred_class")
accuracy_radial_filt <- accuracy_radial_filt$.estimate  # 0.8333333

# AUC
radial_svm_auc_filt <- radial_filt_augmented %>%
    roc_auc(y, .pred_1)
radial_svm_auc_filt <- radial_svm_auc_filt$.estimate  # 0.8814815

# ROC curve
radial_roc_filt <- radial_filt_augmented %>%
    roc_curve(y, .pred_1)
```

Compare the results with the ones before noise removal

```r
accuracies <- data.frame(matrix(nrow = 2, ncol = 3))
accuracies[1, ] <- c(accuracy_linear, accuracy_poly, accuracy_radial)
accuracies[2, ] <- c(accuracy_linear_filt, accuracy_poly_filt,
    accuracy_radial_filt)
colnames(accuracies) <- c("Linear", "Polynomial", "Radial")
rownames(accuracies) <- c("Before", "After")
knitr::kable(accuracies)
```

	Linear	Polynomial	Radial
Before	0.6666667	0.6250000	0.7083333
After	0.7500000	0.7083333	0.8333333

```r
auc <- data.frame(matrix(nrow = 2, ncol = 3))
auc[1, ] <- c(linear_svm_auc, poly_svm_auc, radial_svm_auc)
auc[2, ] <- c(linear_svm_auc_filt, poly_svm_auc_filt, radial_svm_auc_filt)
colnames(auc) <- c("Linear", "Polynomial", "Radial")
rownames(auc) <- c("Before", "After")
knitr::kable(auc)
```

	Linear	Polynomial	Radial
Before	0.9037037	0.7481481	0.8814815
After	0.8296296	0.9111111	0.8814815

```r
b1 <- autoplot(linaer_roc)
b2 <- autoplot(poly_roc)
b3 <- autoplot(radial_roc)
b1 + b2 + b3
```

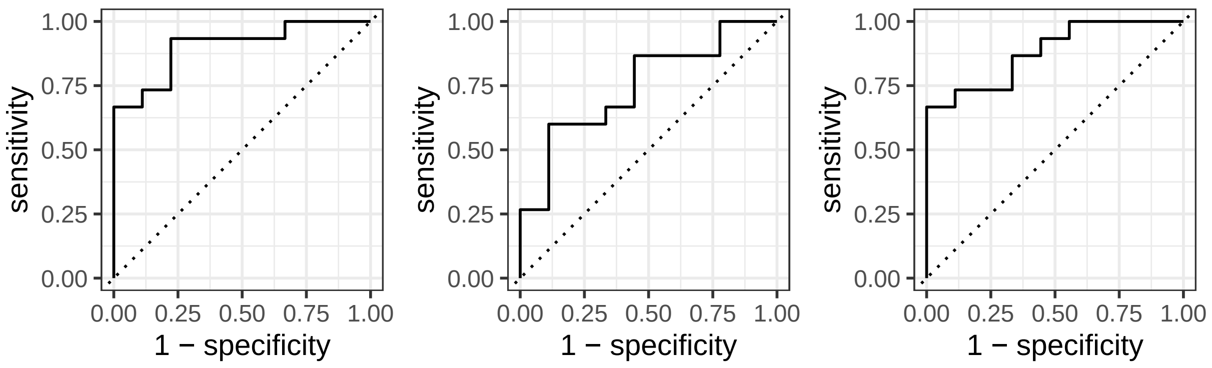

Figure 1: ROC curves before noise removal. From the left to right, we have linear SVM, polynomial SVM and radial SVM.

```r
a1 <- autoplot(linaer_roc_filt)
a2 <- autoplot(poly_roc_filt)
a3 <- autoplot(radial_roc_filt)
a1 + a2 + a3
```

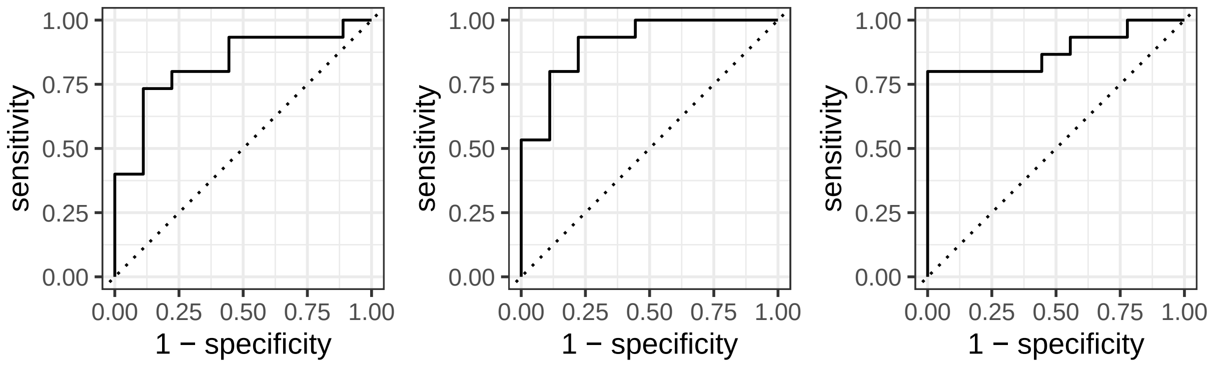

Figure 2: ROC curves after noise removal. From the left to right, we have linear SVM, polynomial SVM and radial SVM.

Conclusions

As we already mentioned, before the noise removal, the radial kernel is the SVM with highest accuracy. However, it is the linear kernel the one with better AUC value. Instead, after the noise removal, the accuracies improved. In particular, the radial kernel is still the one with better accuracy, but the one with higher AUC is the polynomial kernel.

In general, these findings align with the notion that machine learning models encounter difficulties when confronted with extremely multidimensional data. In this case study, it is observed that the low variance predictors lack representativeness. Their inclusion in the dataset only contributes to the introduction of noise. The table reporting all the AUC indicates that these unrelated characteristics have led to a confusion of the models. By considering solely the most informative variables, we were able to mitigate the impact of irrelevant information within the dataset. Consequently, we obtained models that were less complex but exhibited enhanced performance, thereby enabling effective generalization on the training data.

Finally, to improve our analysis, it would have been better to have a larger pool of parameters to test which one is the optimal hyperparameter.