Statistical Learning, Homework #3

Anastasia Santo

28/05/2023

The data set that will be analyzed contains data from 79 leukemia patients belonging to two subgroups : patients with a chromosomal translocation (1) and cytogenetically normal patients (-1). For each patient a Sample ID and expression data for 2000 genes are reported, and an additional column containing the reference subgroup. The number of patients with a chromosomal translocation is 37; the remaining 42 are cytogenetically normal. The dataset contains no missing values so we can proceed performing a supervised analysis for prediction of the subgroups using support vector machines.

```
library("ISLR2")
library("e1071")
library("caret")
```

```
## Caricamento del pacchetto richiesto: ggplot2

## Caricamento del pacchetto richiesto: lattice
```

```
gene<-read.csv("C:/Users/santo/OneDrive/Desktop/Regression and Classification models/gene_expr.tsv",sep
#dim(gene)
gene<-na.omit(gene)
#dim(gene[gene$y=="-1",])
```

Support Vector Machines

Since SVM creates a decision boundary which makes the distinction in this case between two classes, how to draw or determine the decision boundary is the most critical part in SVM algorithms. When the data points in different classes are linearly separable, it is an easy task to draw a decision boundary, but real data is noisy and not linearly separable in most cases. First of all different type of `Kernel` should be evaluated :

- `kernel="linear"`: enable us to deal with data having linear (or linear-like) class boundaries.

To fit an SVM on data with non-linear class boundary:

- `kernel="radial"`: set the value of γ for the radial basis kernel changing the `gamma` argument;
- `kernel="polynomial"`: set the polynomial degree changing the `degree` argument, the `gamma` and the `coef0`.

For each type of kernel an important parameter to choose is C : The value C bounds the sum of the slack variables that allow individual observations to be on the wrong side of the margin of the hyperplane. For `C>0` no more than C observations can be on the wrong side of the hyperplane; C controls the bias-variance trade-off : if C is small we will have a highly fit to the data, so a low bias, while when C is larger we have a classifier that is more biased but may have lower variance. Using the `svm()` function we can tune the parameter C through the `cost` value, which is inversely proportional to the C value we mentioned earlier, and which represents the cost of a violation to the margin. * small `cost` -> wide margins, many support vectors on the margins; * large `cost` -> narrow margins, few support vectors on the margins.

First of all the dataset is divided into a training set (80%) and a test set (20%), since only the training set will be used to do the hyperparameters tuning and the test set will be used in the last part to evaluate each model. Since the dataset is quite balanced there is no need to balance the sampling.We first encode the response as a factor variable, otherwise `svm()` will perform a support vector regression. To improve model performance and convergence speed feature normalization is done. In fact if one feature has very large values, it will dominate the other features when calculating the distance.

```
gene1<-gene[,-c(1, 2002)]
genesc<-as.data.frame(scale(gene1))
gene[,"y"]<-as.factor(gene[,"y"])
gene2<-gene
gene2[,c(seq(2:2001))]<-genesc
gene2[,1]<-gene[,1]
gene2[,"y"]<-as.factor(gene2[,"y"])
set.seed(1234)
n <- nrow(gene2)
train_size <- 0.8
train <- sample(n, train_size * n)
genetrain<-gene2[train,]
genetrain[,"y"]<-as.factor(genetrain[,"y"])
genetest<-gene2[-train,]
genetest[,"y"]<-as.factor(genetest[,"y"])
```

The training set will be used within the `tune()` function to evaluate the best `cost` and other kernel parameters values, and the latter will then be applied in the `svm()` function to create the best SVM model that will then be tested on the test set. The `tune()` function tunes hyperparameters of statistical methods using a grid search over supplied parameter ranges, performing a 10-fold cross validation on the training set for different cost values that are provided to it. A list of cost parameters to be tested is provided here :

```
cost_range<-c(0.001, 0.01, 0.1, 1, 5, 10, 100)
```

The first tuning will be done using an SVM model, setting kernel=linear , using the training set, and setting in ranges the list for costs values. As default a 10-fold cross validation will be done to compute the cv-error for each choice of cost value.

```
set.seed(1234)
tune.outl <- tune(svm, y ~ ., data=genetrain, kernel="linear", ranges=list(cost=cost_range))
#summary(tune.outl)
opt.costl <- tune.outl$best.parameters$cost
```

The best value for the cost is 0.01, after this value the cv.error remains the same. The lower value of dispersion is reached instead for cost = 0.001. Now the same procedure will be performed, but this time using kernel=polynomial. In this case the parameters that need tuning are cost, degree, gamma and coef_0. The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. low gamma –> points far away from plausible seperation line are considered in calculation for the seperation line. high gamma –> points close to plausible line are considered in calculation. The `degree` defines the polynomial degree, while `coef_0` allow you to adjust the independent term in your kernel function. Here the function `tune.svm()` is used, and it gives in output the best values for each parameter. Since the dataset already have a high number of features, I choose polynomial degree until 4, because of the high number of features, and therefore of the dimension of the linear hyperplane.

```
gamma_range <- c(0.0001,0.001,0.01, 0.01, 1, 5, 10)
degree_range<-c(2,3,4)
coef_0_range<-c(0.001,0.01, 1)
set.seed(1234)
tune.outp <- tune(svm, y ~ ., data=genetrain, kernel="polynomial", ranges=list(cost=cost_range,
#summary(tune.outp)
opt.costp<-tune.outp$best.parameters$cost
opt.gammap<-tune.outp$best.parameters$gamma
opt.degree<-tune.outp$best.parameters$degree
```

The best value for the cost is 5, the best value for gamma is 0,001, the best value for coef_01 is 0 as default, and the best degree value is 3. Now I will perform the same procedure, but this time using kernel=radial, the parameters which need tuning are cost and gamma.

```
set.seed(1234)
tune.outr <- tune(svm, y ~ ., data=genetrain, kernel="radial", ranges=list(cost=cost_range,gamma=gamma_r
#summary(tune.outr)
opt.costr<-tune.outr$best.parameters$cost
opt.gammar<-tune.outr$best.parameters$gamma
```

The best value for cost in this case is 10, while the best value for gamma is 0.001. Now that the best parameters for each kernel have been evaluated, we can model different SVM model with the three different kernel on the training data, using the best parameters of cost, gamma, degree, and coef_0. Since y is been `factored` and a classification need to be performed, I set the `type` choice to `C-classification`. Per default, data are scaled internally (both x and y variables) to zero mean and unit variance, so scale=FALSE is selected, since x had already been scaled. While fitting the model `decision.values=TRUE` is selected, in order to obtain the fitted values of the SVM model.

```
svmfitl <- svm(y ~ ., data=genetrain, kernel="linear", cost=opt.costl,scale = FALSE,type="C-classificat
                      decision.values=TRUE)
svmfitp <- svm(y ~ ., data=genetrain, kernel="polynomial", cost=opt.costp, gamma=opt.gammap, degree=opt
                      coef_0= opt.coefp,scale=FALSE,type="C-classification",decision.values=TRUE)
svmfitr <- svm(y ~ ., data=genetrain, kernel="radial", cost=opt.costr,
                      gamma=opt.gammar,scale=FALSE,type="C-classification",decision.values=TRUE)
```

NUMBER OF SUPPORT VECTORS : * `kernel="linear" SVM`: 53 * `kernel="polynomial (3)" SVM`: 21 * `kernel="radial" SVM`: 61 The radial kernel SVM has the wider margins since it has the highest number of support vectors, while the polynomial has the tighter ones. Now predictions on test set are computed using the different models. Confusion matrix are shown, and the decision values will be extrapolated using the parameter `decision.values=TRUE` and then used to built ROC curves.

```
pr1<-predict(svmfitl, genetest, decision.values=TRUE)
pr2<-predict(svmfitp, genetest, decision.values=TRUE)
pr3<-predict(svmfitr, genetest, decision.values=TRUE)
table(true=genetest$y, pred=pr1)
```

```
##     pred
## true -1 1
##   -1  8 0
##   1   2 6
```

```
table(true=genetest$y, pred=pr2)
```

```
##     pred
## true -1 1
##   -1  7 1
##   1   3 5
```

```
table(true=genetest$y, pred=pr3)
```

```
##     pred
## true -1 1
##   -1  5 3
##   1   5 3
```

```
fitl<-attributes(pr1)$decision.values
fitp<-attributes(pr2)$decision.values
fitr<-attributes(pr3)$decision.values
```

The ROC curves are a useful diagnostic tool for interpreting the performance of a binary classifier: they summarize in a plot the trade-off between the *true positive rate* and the *false positive rate* for different probability thresholds. This is the utility function for drawing a ROC curve using the `ROCR` R library. The inputs are a vector of numerical scores for each observations (`pred`), and a vector of true labels (`truth`).

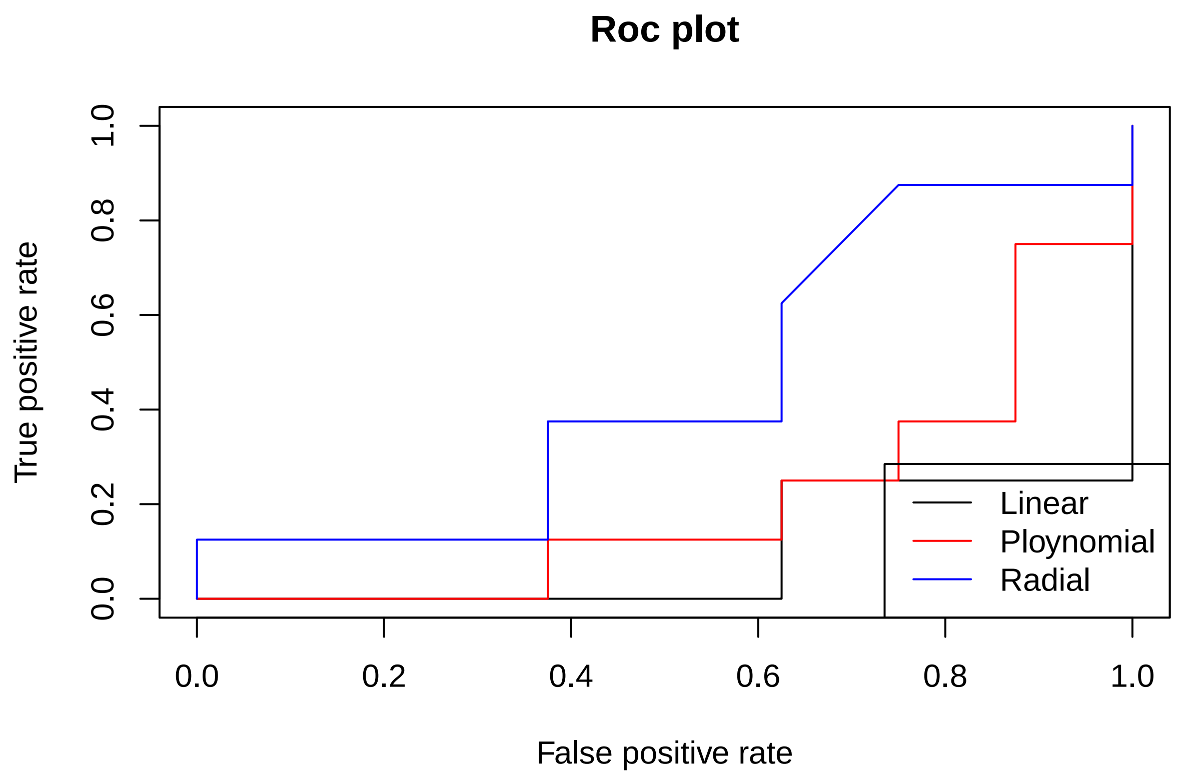
```
library(ROCR)

rocplot <- function(pred, truth, ...){

    predob <- prediction(pred, truth)
    perf <- performance(predob, "tpr", "fpr")
    plot(perf, ...)
}
```

Here the ROC curve for each fit is plotted. The predicted label $\hat{y}$ is given by $\hat{y} = sign(\hat{\beta}_0 + \beta^T X)$. In equivalent terms, $\hat{y} = 0$ if $\hat{\beta}_0 + \beta^T X < 0$, $\hat{y} = 1$ otherwise. Using the ROC curve the positive detection rate can be controlled by introducing a threshold ϵ so that $\hat{y} = 0$ if $\hat{\beta}_0 + \beta^T X < \epsilon$, $\hat{y} = 1$ otherwise. So the ROC curve is computed by varying this ϵ. At this point a comparison can be made through the ROC curve corresponding to all the models.

```
rocplot(fitl, genetest$y, main="Roc plot")
rocplot(fitp, genetest$y, add=T, col="red")
rocplot(fitr, genetest$y, add=T, col="blue")
legend("bottomright", legend=c("Linear", "Ploynomial","Radial"), col=c("black", "red","blue"), lty=1)
```



Looking at the ROC curve it seems that the AUC for the blue curve is greater than the AUC for the other ones, so it tells us that the SVM with radial kernel performs better than the other models. The better performance in the roc curve is also due to the fact that the svm model with radial kernel has wider margins than the others (61 support vectors), so it may fit better unseen data, avoiding overfitting. To conclude, it can be said that the SVM model with radial kernel performs best in classification using all the genes in the dataset. By decreasing the number of genes and using only those showing higher differential expression, the results could show better performance of the model.