R Tutorial for Statistical Learning

Qidi Peng

16: Support vector machines

Code	Comments	Results

#Support vector classifier.
install.packages("e1071");
library(e1071);
	The package "e1071" includes built-in functions for support vector machines.	

#Generate 2 classes of observations with linear boundary. set.seed (1);		
x=matrix (rnorm (20*2) , ncol =2);	First generate a sample of 20 independent 2-dimensional normal vectors.	
y=c(rep (-1,10) , rep (1 ,10));	Set a set of responses y (for labeling rows of x). By the support vector classifier approach, y takes values either -1 or 1.	
x[y==1 ,]= x[y==1,] + 1;	Add 1 to each of the second half rows of x. Then one has 2 classes of normal vectors: one is the first 10 rows, which is mean zero, the other one is the remaining 10 rows of x, which is mean (1,1).	
plot(x, col =(3-y));	Visualize the 2 classes of observations, using distinct colors. (3-y) is the label of colors. This step is necessary to check whether the classes are well separated. We see here the data are not well separated. But the boundary is almost linear.	
dat=data.frame(x=x, y=as.factor (y));	Create data frame. The response y must be factors.	
svmfit=svm(y~.,data=dat,kernel="linear",cost=10,scale=FALSE);	Run a support vector classifier. Scale=FALSE means we don't let the data be normalized. cost=10 specifies a tuning parameter, it shows the length of margins, smaller is cost, wider are the margins.	
plot(svmfit , dat);	Visualize the linear classifier. The region of feature space that will be assigned to the −1 class is shown in light blue, and the region that will be assigned to the +1 class is shown in purple. We see the margins are soft: they allow for some violations of observations.	

svmfit$index;
 	
summary (svmfit);
 	
#To see the property of the argument “cost”, we take an extreme case.
svmfit=svm(y~.,data=dat,kernel="linear",cost=0.1,scale=FALSE);
plot(svmfit , dat);
svmfit$index;
 	
#To choose the best “cost”, we perform a cross-validation.
set.seed(1);
tune.out=tune(svm,y~.,data=dat,kernel ="linear",ranges=list(cost=c(0.001,0.01,0.1,1,5,10,100)));
summary (tune.out);
 	
bestmod =tune.out$best.model;
summary(bestmod);
 	
#Prediction.
xtest=matrix (rnorm (20*2) , ncol =2);
ytest=sample (c(-1,1) , 20, rep=TRUE);
xtest[ytest ==1 ,]= xtest[ytest ==1,] + 1;
testdat =data.frame (x=xtest , y=as.factor (ytest));
 	
ypred=predict (bestmod ,testdat);
 	
table(predict =ypred , truth= testdat$y);
	Check the indices of support vectors. 1 to 10 belong to Class -1, 11-20 belong to Class 1.
	Print the summary of this model.
	
	Take cost=0.1, which is quite small.
	Plot the classifier.
	Print the labels of support vectors. We see the margins are quite wide.
	
	Use function tune(), we select the best cost among some candidate values.
	A summary tells you the best choice of cost is 0.1.
	
	This is another way to show the best cost, by outputting the variable best.model.
	
	In order to test our decision boundary, we create another 20 test data.
	
	Use predict() to classify the test data, the decision boundary is provided by the model “bestmod”, the one with cost=0.1.
	The confusion matrix shows only 1 test data is mistakenly classified by the decision boundary.
		[1] 1 2 5 7 14 16 17
		
		[1] 1 2 3 4 5 7 9 10 12 13 14 15 16 17 18 20
		
		- best performance: 0.1
		
		truth
		predict -1 1
		-1 11 1
		1 0 8

#Support vector machine.
#Generate 2 classes of non-linear boundary observations.
set.seed (1);
x=matrix (rnorm (200*2) , ncol =2);
y=c(rep (1 ,150) ,rep (2 ,50));
 	
x[1:100 ,]=x[1:100 ,]+2;
x[101:150 ,]= x[101:150 ,] -2;
dat=data.frame(x=x,y=as.factor (y));
 	
plot(x, col=y);
train=sample (200 ,100);
	
	Generate 200 normal observations of dimension 2.
	Set labels of classes. The first 150 data go to Class 1, the remaining 50 data go to Class 2.
	Change the first 100 data in Class 1.
	Change the values of Class 2.
	Build a data frame.
	
	We see the boundary of the 2 classes is quite irregular.
	Randomly choose 100 observed data for training. The	

svmfit=svm(y~.,data=dat[train ,],kernel="radial",gamma=1,cost=1);
plot(svmfit , dat[train ,]);
summary (svmfit);
	remaining 100 will be testing data.
	Perform a radial kernel support vector machine.
	Visualize the decision boundary.
	Summary of the model outputs.	

#Choose the best values of parameters gamma and cost, using cross-validation.
tune.out=tune(svm,y~.,data=dat[train ,],kernel="radial",ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),gamma=c(0.5,1,2,3,4)));
summary (tune.out);
	Run a cross-validation to choose the best parameters among some candidate values.
	The summary shows the best choice is cost=1,gamma=2.
		- best parameters:
		cost gamma
		1 2

table(true=dat[-train ,"y"],pred=predict(tune.out$best.model,newx=dat[-train,]));	Use the validation set to test the best model (cost=1,gamma=2). We see from the confusion matrix that, 39 over 100 data are mistakenly classified by the best model.	pred true 1 2 1 56 21 2 18 5

#SVM with multiple classes.
#When there are more than 2 classes, the svm() function will automatically perform multi-class #classification using the one-versus-one approach.
 	
#Generate 3 classes of observations.
set.seed (1);
x=rbind(x, matrix (rnorm (50*2) , ncol =2));
y=c(y, rep (0 ,50));
x[y==0 ,2]= x[y==0 ,2]+2;
dat=data.frame(x=x, y=as.factor (y));
par(mfrow =c(1,1));
plot(x,col =(y+1));
svmfit=svm(y~.,data=dat,kernel="radial",cost=10,gamma=1);
plot(svmfit , dat);
 	
help(svm);
	Add 50 rows to the previous observations x.
	Build the third label of class y=0.
	Change the values of the third class.
	Set the data as data frame.
	
	Plot the observations.
	Run a radial kernel support vector machine.
	Illustrate the decision boundary.
	
	We see in the function svm(), the argument kernel has other options as “polynomial” and “sigmoid”. Please try them with your observations.	

17: Unsupervised learning: PCA

Codes	Comments	Results

# Principal component analysis. 	
# We perform PCA to “USArrests” data set. 	
states =row.names(USArrests);
states;
names(USArrests);
	
	The rows of the data set are state names.
	Check the names of the 50 states.
	Check the variables contained in the data set.
		
		[1] "Murder" "Assault" "UrbanPop" "Rape"

pr.out=prcomp(USArrests,scale =TRUE); summary(pr.out);	prcomp() function performs PCA to the data set. Summary of the outputs. We see there are totally 4 components (since p=4). By comparing the proportions of variances, we believe the first 2 principal components are the most important to explain tell the dimension of the data set.	
biplot(pr.out, scale =0);	Visualize the first 2 PCs. From the image we see Virginia has every feature in the middle, California has high rate of rape, Mississippi has high rate of murder, etc.	
pr.out$center; pr.out$scale; pr.out$rotation;	The means of the 4 crimes. The standard deviations of the 4 crimes. The corresponding PC loading vectors.	

pr.out$rotation=-pr.out$rotation;
pr.out$x=-pr.out$x;
biplot (pr.out , scale =0);
	Change the directions of PC.
	
	Let the coordinates be positive. We reproduce the image.	

names(pr.out);	Check all the output variables of pr.out.	

u=(pr.out$sdev^2)/sum(pr.out$sdev^2);
plot(u,xlab="Principal Component",ylab="Proportion of Variance Explained",ylim=c(0,1),type="b");
plot(cumsum(u), xlab=" Principal Component ",ylab ="Cumulative Proportion of Variance Explained ",ylim=c(0,1),type="b");
	We show the cree plot, which explains how much does each PC explain the variance.
	
	The cumulative sum of variances explained by the first PCs. These images will help us to decide how many PC should be considered in a selected model.	

18: Clustering analysis

Codes	Comments	Results
#K-means.		

#Example 1: clustering a numerical data set.
set.seed (2);
x=matrix (rnorm (50*2) , ncol =2);
 	
x[1:25 ,1]=x[1:25 ,1]+3;
x[1:25 ,2]=x[1:25 ,2] -4;
 	
km.out =kmeans (x,2, nstart =20);
 	
km.out$cluster;
 	
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with K=2", xlab ="",ylab="",pch =20,cex =2);
 	
set.seed (4);
km.out =kmeans (x,3, nstart =20);
km.out;
 	
km.out$tot.withinss;
 	
km.out$withinss;
 	
#Example 2: Clustering analysis to image data.
 	
install.packages("ripa");
library(ripa);
install.packages(“jpeg”);
library(jpeg);
img =readJPEG("C:/Users/Peng/Desktop/id.jpg");
 	
plot(imagematrix(img));
dim(img);
ID=kmeans(img,3);
C=matrix(ID$cluster,2004,2672,3);
	Artificially generate 2 groups of data. Each group has 25 observations.
	
	One has a shift of 3;
	The other one has a shift of -4.
	
	Perform a K-means clustering with K=2. nstart is the number of random assignments run for initial step. If nstart=20, R runs 20 times algorithm for 20 different initial clusters, then shoose the best result. So we had better choose a big nstart value.
	The label of cluster for each observation.
	
	Plot the results. The 2 clusters are of distinct colors.
	
	We try K-means with K=3.
	
	Print the results.
	
	The total within-cluster sum of squares, which we seek to minimize by performing K-means clustering.
	
	The individual within-cluster sum-of-squares.
	
	This package allows to load JPEG and convert image to matrix, and convert matrix to image.
	Download the image data id.jpg from Canvas. Convert it into a 3D matrix. The components (fractions) of img correspond to pixels. 0=black, 1=white.
	Convert the matrix img into a picture.
	Check the dimension of img.
	Perform K-mean with K=3s.
	Create a cluster indices matrix
		[1] 97.97927
		
		[1] 2672 2004 3

C=t(C); plot(imagematrix(C/3));	corresponding to each element of img. Transpose C. Simulate the id.jpg, using labels of clusters. The 3-means assigns each value in C to be among 1,2,3, then C/3 is to make sure that the components are all fractions.	

# Hierarchical clustering. 	
hc.complete=hclust(dist(x),method ="complete");
hc.average=hclust(dist(x),method ="average");
hc.single=hclust(dist(x),method ="single");
	These 3 functions perform the same Euclidean distance dist() hierarchical clustering, respectively, they plot the hierarchical clustering dendrogram using complete, single, and average linkage clustering.
	
	In complete-link (or complete linkage) hierarchical clustering, we merge in each step the two clusters whose merger has the smallest diameter (or: the two clusters with the smallest maximum pairwise distance).
	
	In single-link (or single linkage) hierarchical clustering, we merge in each step the two clusters whose two closest members have the smallest distance (or: the two clusters with the smallest minimum pairwise distance).
	
	Average-link (or group average) clustering is a compromise between the sensitivity of complete-link clustering to outliers and the tendency of single-link clustering to form long chains that do not correspond to the intuitive notion of clusters as compact, spherical objects.	

par(mfrow =c(1,3));
plot(hc.complete,main="Complete;Linkage",xlab="",sub="",cex=.9);
plot(hc.average,main ="Average Linkage",xlab="",sub="",cex=.9);
plot(hc.single,main ="Single Linkage",xlab="",sub="",cex=.9);
	Illustrate the 3 dendrograms.	

cutree(hc.complete,2);	Output the cluster labels for 2 subgroups clustering.	
xsc=scale (x);	One usually scales the data set before running hierarchical clustering.	