SY19 - Apprentissage à partir de trois jeux de données

Jingyi HU

Decembre 6, 2017

The goal of this TP is to build classifiers as powerful as possible from three real data sets. I have 3 datasets:

- expression_train.txt, containing the gray levels of an image associated with a facial expression among joy, surprise, sadness, disgust, anger, fear;
- characters_train.txt, consisting of examples from 26 classes corresponding to the 26 letters of the alphabet;
- speech_train.txt, where each observation corresponds to pronunciation of a phoneme among "sh", "dcl", "iy", "aa" and "ao". I will therefore compare the predictive qualities of several models on these datasets.

NB: If I have enough time, I may try a selection of main components by Forward Stepwise Selection. Because the analysis ideas of the 3 datasets are almost same, I will analyze more precisely for the first one. And I will write only several codes for the second and third dataset.

Load of data

```r
data_expressions<-read.table("expressions_train.txt")
data_characters<-read.table("characters_train.txt")
data_parole<-read.table("parole_train.txt")
```

Using methods

1. The dimension reduction methods:

- Principal Component Analysis (PCA)
- Factor Discriminant Analysis (FDA)
- Forward Stepwise Selection
- …

2. Comparison betIen the performances of these different models:

- K nearest neighbors (KNN)
- Linear Discriminant Analysis (LDA) / Quadratic Discriminant Analysis (QDA) / Regularized Discriminant Analysis (QDA) / Naive Bayesian Classifier
- Random Forest (RF)
- Support Vector Machines (SVM)
- Neural Networks (NN)

3. In addition, I also thought about the possibility to proceed by **cross validation** in order to be able to learn the models on more data, and I tried the 2 at the same time (without and with CV) I get almost the **same result**. Indeed, the number of observations available is quite low. As a result, I didn't implement the cv to methods because the results obtained seemed good.
4. I didn't try the **Logistic Regression and Generalized Additive Models (GLM & GAM)**, because this classification problem is not a binary classification. I didn't use logistic regression because the logistic regression method performed by GLM and GAM (which is an extension of GLM) is often less efficient than discriminant analysis (LDA, QDA . . .) for classifications with more than two classes. In addition,the latter is often used to explain the effects of the different predictors, which is not the purpose of our study. As a result, these methods will not be used for the "word" and "characters" data sets either.

1.translation_expression.txt

First analysis

Pre-treatment

By displaying the different images, I also realize that part of each image is totally useless in the context of our analysis, or even risk noises: it is all the black pixels at the bottom of each image. I decide to exclude them from any future analysis.

"Average faces"

To better analyze this database, I tried to average for each expression of images: that is to say, produce six images from the mathematical average of the images of the expression considered.

Methods to use

After studying the dataset, I found that it has **a large number of predictors compared to the number of individuals**. The different predictors are globally very **correlated**. I will therefore have to use dimension reduction methods on this dataset, and then test our different models.

The dimension reduction methods

PCA

I used the PCA using the *prcomp* function of R. In order to retain only a certain number of principal components, I'm interested in the proportion of variance explained by each of them by plotting their explained proportion of variance, and cumulative variance. You can find a pdf named *pc_cumpropvar.pdf*, *pc_propvarexp.pdf* and *pc_propvarexp2.pdf*. In order to test the performance of models on different selections of **Principal Component (PC)** , I composed several dataset, PCi: From the first i PCs, representing i% of explained variance. I also tried to plot our first two main component, hoping to see some interesting pattern. You can find a pdf named *pc_2prcomp.pdf*.

FDA

According to the poly, I used the lda method, and multiplied the matrix **X** of the individuals by the matrix *lda.scaling* of the selected eigenvectors. I applied the FDA two datasets: 1. Raw FDA: dataset containing all of the individuals in the initial dataset 2. PC100 FDA: dataset containing the 100 PCs of all individuals in the initial dataset. According to the result, I found that I have an overfitting result, the error rate is too low, and the classes are clearly too separated. The FDA on PC100 makes me realize that I didn't use the right approach regarding the FDA. Therefore, I tried using **Cross-Validation** 6-folds on FDA, and I have a rough estimate of the FDA test error rate. This time the overfitting finished.

Conclusion:

1. **the use of PCA before FDA application** and **simply using FDA on the raw dataset** both have good results.
2. The choice of the number of PCs is important: a large number of PCs may reproduce our previous ___ overfitting___ problem, while too few of them do not allow a good representativeness of the initial dataset, because they contain a too small proportion of explained variance in this case.

As FDA on the raw dataset works well, I will analyze more precisely by directly using the **FDA on the raw dataset**.

1.1 FDA

```r
data1<-data_expressions
set.seed(2000)
#Generate the trainning data and test data
N<-nrow(data1)
nbTrain=floor(3/4*N)
nbTst=N-nbTrain
trainIdx<-sample(1:N, nbTrain)
train<-data1[trainIdx,]
test<-data1[-trainIdx,]
```

Now we separate the dataset into a test set and trainning set.

```r
#Remove column full of 0
cleanData.train<-train[,colSums(abs(train[,1:ncol(train)-1])) !=0]
cleanData.test<-test[,colSums(abs(test[,1:ncol(test)-1])) !=0]
#FDA (with train data)
library("MASS")
lda_data1<- lda(y~.,data=cleanData.train)
U<-lda_data1$scaling
X<-as.matrix(cleanData.train[,1:ncol(cleanData.train)-1])
Z<-X%*%U
Z<-as.data.frame(Z)
y<-cleanData.train$y#delete useless informtion
train.FDA<-cbind(Z,y)
#Apply FDA on test data
X<-as.matrix(cleanData.test[,1:ncol(cleanData.test)-1])
Z<-X%*%U
Z<-as.data.frame(Z)
y<-cleanData.test$y
test.FDA<-cbind(Z,y)
#plot(train.FDA[,1], train.FDA[,2], col = train.FDA$y)
```

1.2 K nearest neighbors (KNN)

```
library(class)
train.X <- train.FDA[,1:ncol(train.FDA)-1]
train.Y <- train.FDA[,ncol(train.FDA)]
test.X <- test.FDA[,1:ncol(test.FDA)-1]
test.Y <- test.FDA[,ncol(test.FDA)]

nbvoisin <- seq(1,65)
error <- rep(1:65)
for (i in (1:65)){
  knn.pred=knn(train.X,test.X,train.Y,k=i)
  table(knn.pred,test.Y)
  error[i] <- 1 - mean(knn.pred == test.Y)
}
#plot(nbvoisin,error,xlab="Nombre de voisin pris pour apprentissage",ylab="Erreur estimée")
which.min(error)
```

```
## [1] 6
```

```
print(error[6])
```

```
## [1] 0.1851852
```

According to the result, we obtain the minimum error of **0.1851852** when k = 6. The K nearest neighbors method generally has poor results in large dimension since the neighbors are actually very far from each other. But here I have done a dimension reduction, so it has a not bad results and no overfitting.

1.3 Discriminant Analysis:LDA QDA RDA naive Bayesian

1.3.1 LDA

```
lda_data<- lda(y~.,data=train.FDA)
pred<-predict(lda_data,newdata=test.FDA)
table<-table(test.FDA$y,pred$class)
#table
error<-1-sum(diag(table))/nbTst
error
```

```
## [1] 0.1481481
```

```
library(caret)
library(MASS)
folds <- createFolds(y, k = 10, list = TRUE, returnTrain = FALSE)
error_rates_lda <- matrix(nrow = 10, ncol = 1)
for (i in 1:10) {
  train_df_lda <- as.data.frame(X[-folds[[i]],])
  test_df_lda <- as.data.frame(X[folds[[i]],])
  lda_data<- lda(y~.,data=train.FDA)
  lda.pred <- predict(lda_data, newdata = test.FDA)
  error_rates_lda[i, ] <- length(which(as.vector(lda.pred$class) != as.vector(test.FDA$y)))/length(as.v
}
print(mean(as.vector(error_rates_lda)))
```

```
## [1] 0.1481481
```

We get a test error rate with a cross-validation 10-folds is **0.1481481**, the same result with LDA without CV. Linear boundaries therefore give a satisfactory result.
### 1.3.2 QDA

```
qda_data<- qda(y~.,data=train.FDA)
pred<-predict(qda_data,newdata=test.FDA)
table<-table(test.FDA$y,pred$class)
#table
error<-1-sum(diag(table))/nbTst
error
```

```
## [1] 0.4074074
```

We get a very high test error rate: **0.4074074**. This classifier is probably too flexible for our data, so it should be excluded.

1.3.4 Naive Bayesian Classifier

```
library(klaR)
```

```
## Warning: package 'klaR' was built under R version 3.3.3
```

```
naivB_data<-NaiveBayes(y~.,data=train.FDA)
pred<-predict(naivB_data,newdata=test.FDA)
table<-table(test.FDA$y,pred$class)
# table
error<-1-sum(diag(table))/nbTst
error
```

```
## [1] 0.2592593
```

We get an error rate of **0.2592593** is therefore worse than that obtained with the LDA (0.14).

1.3.5 Regulated discriminant analysis

```
rda_data <- rda(y~.,data=train.FDA, crossval = TRUE)
pred<-predict(rda_data,newdata=test.FDA)
table<-table(test.FDA$y,pred$class)
#table
error<-1-sum(diag(table))/nbTst
error
```

```
## [1] 0.1851852
```

The error rate is **0.1851852**.

1.4 Mixture Discriminant Analysis

```
library(mclust)
ind_y = 6
MclustDa_data <- MclustDA(train.FDA[,1:ind_y-1],train.FDA[,ind_y])
#general covariance structure selected by BIC
summary(MclustDa_data, newdata = test.FDA[,1:ind_y-1], newclass = test.FDA[,ind_y])
```

Despite a zero learning error(overfitting), the test error is extremely high (0.4074074).

1.5 Tree

1.5.1 Decision tree

```r
library(tree)
#Full tree
tree_data = tree(as.factor(y)~., train.FDA)
plot(tree_data)
text(tree_data, pretty = 0)
#Cross validation
size<-cv.tree(tree_data)$size
DEV<-rep(0, length(size))
for (i in (1:10))
{
  cv_data = cv.tree(tree_data)
  DEV<-DEV+cv_data$dev
}
DEV <- DEV/10
plot(cv_data$size, DEV, type = 'b')
#Pruning
prune_data = prune.tree(tree_data, best = 7)
plot(prune_data)
text(prune_data, pretty = 0)
#Test Error
y_pred = predict(prune_data, newdata = test.FDA, type = 'class')
table<-table(y_pred, test.FDA$y)
#table
error<-1-sum(diag(table))/nbTst
error
```

The error obtained is **0.4444444**. It is not a good result. We will try to improve it by using Bagging and Random Forest methods.

1.5.2 Bagging

```r
library(randomForest)
#m = p = 5
bag_data = randomForest(y~., data=train.FDA, mtry=5)
ypred = predict(bag_data, newdata=test.FDA, type = 'response')
table<-table(ypred, test.FDA$y)
error<-1-sum(diag(table))/nbTst
error
```

We have the error rate **0.3333333**.

1.5.3 Random Forest

```r
#m = sqrt(p)
rdForest_data = randomForest(y~., data=train.FDA,mtry=3)
ypred = predict(rdForest_data, newdata=test.FDA, type = 'response')
table<-table(ypred, test.FDA$y)
```

```
error<-1-sum(diag(table))/nbTst
error
```

We have the error rate **0.2962963**. The errors obtained by bagging and random forest are lower than that obtained initially by the tree pruned.

1.6 Support Vector Machine

```
library(e1071)
tune.out = tune(svm, y~., data = train.FDA, kernel = "linear",
                range = list(cost=c(0.01, 0.1, 1, 10, 100), gamma = c(0.1, 1, 10)))
summary(tune.out)
svm_data<-svm(y~., data = train.FDA, kernel = "linear", gamma = 0.1, cost = 1)
ypred = predict(svm_data, newdata=test.FDA)
table<-table(ypred, test.FDA$y)
#table
error<-1-sum(diag(table))/nbTst
error
```

If we use the kernel = **linear**, we obtain the error **0.1851852**, and with the kernel = **radial**, we have **0.2222222**.

1.7 Neural Networks (NN)

```
library('nnet')
train.FDA$y = factor(train.FDA$y)
test.FDA$y = factor(test.FDA$y)
model.nnet = nnet(y ~ ., data=train.FDA, size=2, MaxNWts = 20000)
model.nnet.predicted = predict(model.nnet, test.FDA, type="class")
table<-table(model.nnet.predicted, test.FDA$y)
error<-1-sum(diag(table))/nbTst
error
# perfMeasure(model.nnet.predicted, test.FDA$y)
```

The error here is too large and we won't take it.
The classifier **LDA** is therefore the best for "expressions".

2 Données characters_train.txt

```
data2 = data_characters
dim(data2)
set.seed(1)
```

Since I have a large number of observations (10000) and a relatively small number of variables (17), I don't think it necessary to carry out a size reduction. I simply share the data into a test set and a trainning set.

```
#Generate train data and test data
N<-nrow(data2)
nTrain=floor(3/4*N)
nTest=N-nTrain
train.num<-sample(1:N, nTrain)
```

```
train<-data2[train.num,]
test<-data2[-train.num,]
```

2.1 K nearest neighbors(KNN)

```
  knn.pred=knn(train.X,test.X,train.Y,k=i)
  table(knn.pred,test.Y)
```

I obtain, by cross validation, a minimal error of 0.0760. However, it corresponds to only one neighbor. and I fall into **overfitting**.Here I take the k = sqrt(N).

2.2 Analyze Discriminant

2.2.1 LDA

```
lda_data<- lda(Y~.,data=train)
pred<-predict(lda_data,newdata=test)
```

I get an error rate of **0.3056**. A classifier with a linear boundary does not seem appropriate.

2.2.2 QDA

```
qda_data<- qda(Y~.,data=train)
pred<-predict(qda_data,newdata=test)
```

I obtain an error rate of **0.1224**. The QDA gives good results. This is predictable since I have **a large number of observations**.

2.2.3 Naive Bayesian Classifier

```
naivB_data<-NaiveBayes(Y~.,data=train)
pred<-predict(naivB_data,newdata=test)
```

I get an error rate of **0.3636**. The latter is therefore worse than the one obtained with the QDA (0.1304).

2.2.4 Regulated discriminant analysis

```
rda_data <- rda(Y~.,data=train, crossval = TRUE)
pred<-predict(rda_data,newdata=test)
```

We have a test error rate **0.122** which is relatively close to the one obtained with QDA. We can thus think that the regularization favored a matrix of covariance peculiar to each class.

2.3 Mixture Discriminant Analysis

```r
MclustDa_data <- MclustDA(train[,(indice_y+1):ncol(train)],train[,indice_y])
#general covariance structure selected by BIC
summary(MclustDa_data, newdata = test[,(indice_y+1):ncol(train)], newclass = test[,indice_y])
```

I get good results here training error = **0.02666667** and test error = **0.0828**. This classifier is therefore better than that obtained by QDA(0.1304). I can therefore deduce that the distribution of data within classes is closer to a **mixture of Gausiennes** than to one.

2.5 Tree

2.5.1 Decision tree

```r
library(tree)
tree_data2 = tree(as.factor(Y)~., train)
size<-cv.tree(tree_data2)$size
DEV<-rep(0, length(size))
for (i in (1:10))
{
  cv_data2 = cv.tree(tree_data2)
  DEV<-DEV+cv_data2$dev
}
DEV <- DEV/10
plot(cv_data2$size, DEV, type = 'b')
prune_data2 = prune.tree(tree_data2, best = 17)
y_pred = predict(prune_data2, newdata = test, type = 'class')
table<-table(y_pred, test$Y)
error<-1-sum(diag(table))/nTest
error
```

I use the cross-validation and I get the smallest error for an untagged tree: so I have probably over-learning. This is confirmed by a high test error: **0.616**. This error is therefore not acceptable.

2.5.2 Bagging

```r
bag_data2 = randomForest(Y~., data = train, mtry = 17)
ypred = predict(bag_data2, newdata = test, type = 'response')
table<-table(ypred, test$Y)
error<-1-sum(diag(table))/nTest
error
```

I obtain the **0.0844** test error. I use mtry = p = 5 by uing bagging. It is therefore much better than that obtained previously.

2.5.3 Random Forest

```r
rdForest_data = randomForest(Y~., data = train, mtry = 4)
ypred = predict(rdForest_data, newdata = test, type = 'response')
table<-table(ypred, test$Y)
error<-1-sum(diag(table))/nTest
error
```

The test error is **0.0532**.It is the best result so far(with the minimum test error).

2.6 Neural Networks(NN)

```
model.nnet = nnet(train.Y ~ ., data=train.X, size=2, MaxNWts = 20000)
model.nnet.predicted = predict(model.nnet, test.X, type="class")
table<-table(model.nnet.predicted, test.Y)
error<-1-sum(diag(table))/nTest
error
# perfMeasure(model.nnet.predicted, test.FDA$y)
```

The error is 0.19541.

For the dataset "characters" the best classfier is therefore **Random Forest**.

3 Données parole_train.txt

The dataset has 257 variables. It seems that we need to do a dimension reduction. To be sure, I compared the test errors of an estimated model with and without FDA. The error with FDA is significantly smaller than the error without FDA, so I choose to apply the FDA for all the methods.

```
data3<-data_parole
set.seed(1000)
#Generate train data and test data
N<-nrow(data3);nTrain=floor(3/4*N);nTst=N-nTrain;trainIdx<-sample(1:N, nTrain)
train<-data3[trainIdx,];test<-data3[-trainIdx,]
#FDA (with train data)
lda_data<- lda(y~.,data=train);S<-lda_data$scaling
X<-as.matrix(train[,1:ncol(train)-1]);Z<-X%*%S
Z<-as.data.frame(Z);y<-train$y;train_FDA<-cbind(Z,y)
#Apply FDA on test data
X<-as.matrix(test[,1:ncol(test)-1])
Z<-X%*%S;Z<-as.data.frame(Z)
y<-test$y;test_FDA<-cbind(Z,y)
```

3.1 K nearest neighbors(KNN)

```
  knn.pred=knn(train.X,test.X,train.Y,k=i)
  table(knn.pred,test.Y)
```

I obtain the minimum error of *0.07460036* when k = 17.

3.2 Discriminant Analysis

3.2.1 LDA

```
lda_data3<- lda(y~.,data=train_FDA)
pred<-predict(lda_data3,newdata=test_FDA)
```

I get an error rate of **0.07282416**. Linear boundaries therefore gives a good result.

3.2.2 QDA

```
qda_data3<- qda(y~.,data=train_FDA)
pred<-predict(qda_data3,newdata=test_FDA)
```

I get an error rate of **0.07104796**.

3.2.3 Naive Bayesian Classifier

```
NB_data3<-NaiveBayes(y~.,data=train_FDA)
pred<-predict(NB_data3,newdata=test_FDA)
```

The error rate is **0.06749556**.

3.2.4 Regulated discriminant analysis

```
rda_data3 <- rda(y~.,data=train_FDA, crossval = TRUE)
pred<-predict(rda_data3,newdata=test_FDA)
```

The error rate is **0.06749556**, it's the same with that of Naive Bayesian Classifier, but I don't have the same table. The different discriminant analysis methods thus give similar test errors. These approachs seem to work well on this data.

3.3 Mixture Discriminant Analysis

```
MclustDa_data3 <- MclustDA(train_FDA[,1:indice_y-1],train_FDA[,indice_y])
#general covariance structure selected by BIC
summary(MclustDa_data3, newdata = test_FDA[,1:indice_y-1], newclass = test_FDA[,indice_y])
```

The training error is **0.04682869**, test error is **0.07815275**. This model also gives good result.

3.4 Tree

3.4.1 Decision tree

```
tree_data3 = tree(as.factor(y)~., train_FDA)
#Pruning
prune_data3 = prune.tree(tree_data3, best = 6)
y_pred = predict(prune_data3, newdata = test_FDA, type = 'class')
table<-table(y_pred, test_FDA$y)
```

The resulting test error is **0.07992895**. It is rather small and is obtained without pruning for 6 leaves. This is probably related to the fact that a dimension reduction has been performed, which limits the complexity of the initial tree.

3.4.2 Bagging

```
bag_data = randomForest(y~., data=train_FDA, mtry = 4)
ypred = predict(bag_data, newdata=test_FDA, type = 'response')
```

The error is **0.08703375**.

3.4.3 Random Forest

```
rdForest_data = randomForest(y~., data=train_FDA,mtry = 2)
ypred = predict(rdForest_data, newdata=test_FDA, type = 'response')
```

I obtain the error **0.08880995**. Here I take mtry = sqrt(p).

3.5 Support Vector Machine

```
tune.out = tune(svm, y~., data = train_FDA, kernel = "radial",
                range = list(cost=c(0.01, 0.1, 1, 10, 100), gamma = c(0.1, 1, 10)))
svm_data<-svm(y~., data = train_FDA, kernel = "radial", gamma = 0.1, cost = 1)
ypred = predict(svm_data, newdata=test_FDA)
```

I have the error **0.06927176** with a cost 0.1 and a gamma 0.1.

3.6 Neural Networks(NN)

```
model.nnet = nnet(y ~ ., data=train_FDA, size=2, MaxNWts = 20000)
model.nnet.predicted = predict(model.nnet, test_FDA, type="class")
```

Here we have the error 0.1740675.

The **naive Bayesian** classifier is therefore the best for the dataset "word". Besides, I found that all methods have performed well on this dataset.

Conclusion

- dataset expression : LDA
- dataset character : Random Forest
- dataset parole : Naive Bayesian