SY19 Projet TP7

Part I Données astronomiques

The purpose of Exercise I is to train a model to classify the objets astronomiques given into three type: “star”, “galaxy” and “quasar”. We start with loading the file CSV of data “astronomy_train”.

Preparation

After checking data by the function summary, we verify that there are 5000 observations. We have 17 predictors and a response variable, nameed “class”, without any missing data, so we didn’t have to consider strategies to replace or remove missing values. First of all, we check the predictors. We find that there are constant zero variance predictors, “objid” and “rerun”. All the obeservations have the same value on these two predictors. They don’t have any information useful, and they break some models we want to fit to the data such as LDA. So we choose to delete these two.

```
astro<-astro[,-which(names(astro)=='objid')]; astro<-astro[,-which(names(astro)=='rerun')]
```

Then, we check the the type of the predictors and the response. We find “camcol” and the response “class” are qualitative. The rest is quantative. Then we transform the “class” into factor because that is the way R treat qualitative response in many algorithms. And we transform “run” and “camcol” into dummy variables.

```
astro.raw<-astro
astro<-dummy_cols(astro, select_columns = "camcol"); astro<-astro[,-which(names(astro)=='camcol')];
astro$class<-as.factor(astro$class)
names(astro)
```

```
##  [1] "ra"        "dec"       "u"         "g"         "r"
##  [6] "i"         "z"         "run"       "field"     "specobjid"
## [11] "class"     "redshift"  "plate"     "mjd"       "fiberid"
## [16] "camcol_2"  "camcol_6"  "camcol_1"  "camcol_4"  "camcol_3"
## [21] "camcol_5"
```

We take a quick look on the response, we find the “QSO” is much fewer than “GALAXY” and “STAR”. We need to avoid that we have very small number of “QSO” in testing set. So we decide to split the data evenly by the “class”.

```
summary(data_train$class)
```

```
## GALAXY    QSO   STAR
##   1679    278   1375
```

```
summary(data_test$class)
```

```
## GALAXY    QSO   STAR
##    840    140    688
```

LDA, QDA, Multinomial logistic regression, Naive Bayes

After spliting the data, we begin fitting the data. We decide to try LDA, QDA, Multinomial logistic regression, Naive Bayes. Because these algorithms usually have good performance on the classification problems. Before starting, we create a function to choose the best formular for each algorithme. All these 4 classifiers use linear method, so we can use “regsubsets” to select subset. In the function ‘model.construct’ whose codes are not shown here, we use “Forward Stepwise Selection(FSS)” to choose the predictors which have strong influence in classifiying. We create 15 formulas for each algorithm because we have 15 predictors. The first fomula has one best predictor, the second has 2 best predictors, etc, which makes sure that we have tried all kinds of formulas by the number of predictors. The things that we need to pay attention is that R2 is designed for the problem regression. To get the best predictors, we need to use AIC which add penalty on the number of the predictors. The lower AIC is, the better the model is. Aftering having 15 formulars, we use "K-fold Cross Validation" to train these formulars, and return the best formula with the smallest error of classification for the algorithme given.The code will not be shown here which is too long.

With model.construct function, we can apply these algorithme on our data. For example, we use QDA on our data.

```
model.qda<-model.construct(data_train, "QDA", ntrain)
lm.qda<-qda(as.Formula(model.qda), data=data_train)
pred.qda<-predict(lm.qda, newdata=data_test)
perf <-table(data_test$class,pred.qda$class)
error.qda<-1-sum(diag(perf))/(n - ntrain)
```

```
## Warning in leaps.setup(x, y, wt = wt, nbest = nbest, nvmax = nvmax,
## force.in = force.in, : 1 linear dependencies found
```

The prediction errors of these 4 kinds of models with their best models.

```
##          LDA        QDA Multinomial_logistic_regression Naive_Bayes
## 1 0.08333333 0.01678657                       0.1918465   0.0263789
```

We find out that QDA has a very impressive performance in this classification problem. QDA is the most general model with covariance matrices not equaled. Usually, it does not always have the best performances, because it has the biggest number of parameters, and LDA is more stable than QDA because LDA assumes that covariance matrices are equaled parameters. But in our case, we have 5000 observations, so we can ignore the difference of the covariance matrices. The performance of Naive Bayes classifer is better than LDA but worse than QDA. It makes sense because it assumes that covariance matrices are different but diagonal. Naive Bayes classifier is not the best one, but still is very powerful. Different from LDA,QDA,Naive Bayes which use the full likelihood by applying Bayes' theorem, Multinomial logistic regression uses the conditional likelihood based on the conditional probabilities.

k-nearest-neighbor

Since the decision boundaries are non-linear, we decide to use k-nearest-neighbor classifier. Because this classifier is a nonparametric estimation whih means it doesn't make any assumption on the boudaries. It's good at solving classification problem with non-linear decision boundaries. Now we start to use k nearest neighbors model to fit the data.

```
K<-5; CV.KNN<-rep(0,30)
folds <- sample(1:K, 2*ntrain/3, replace = T)
#We will test the parameter K within the range of 1 to 30
for (i in 1:30){
  for (j in 1:K){
    data.cv.train<-data_train[folds != j,]
    data.cv.test<-data_train[folds == j,]
    fit_pre = knn(data.cv.train[, -which(names(data.cv.train)=="class")],
                  data.cv.test[, -which(names(data.cv.test)=="class")],
                  cl = data.cv.train[, which(names(data.cv.train)=="class")], k = i)
    perf.knn<-table(data.cv.test$class, fit_pre)
    CV.KNN[i]<-CV.KNN[i]+1-sum(diag(perf.knn))/(ntrain/5)}
  CV.KNN[i]<-CV.KNN[i]/5}
```

```r
K.best<-which.min(CV.KNN)
which.min(CV.KNN)
```

```
## [1] 30
```

Since we already have the model of KNN with the best K which is shown above, the prediction error is as below:

```
## [1] 0.1846523
```

The error of misclassification is really high. It isn't what we expect. We try to find the reason of the bad performance. We believe it's curse of dimensionality. We have 20 predictors(after trainsforming "camcol" into dummy variables) which means 20 dimensions. All the obeservation can be far away from each other in high dimension. This causes the bad performance. Usually, KNN is good when number of the predictors p < 5 and number of the examples N is big.

Decision tree

Trees can be displayed graphically, and are easily interpreted, and trees can easily handle qualitative predictors without the need to create dummy variables. Unfortunately, the trees we make have high variances, they generally do not have the same level of predictive accuracy as some of the other modern classification approaches. So we decide to use "Bagging" and "randomForest" directly. Bagging sets m = p. In this exercise, p = 15. So we need to test the m which is around 15 with Cross-Validation. Random Forest sets m = sqrt(p). So we need to test m which is around 4 with Cross-Validation.

```r
astro.raw$camcol<-as.factor(astro.raw$camcol);astro.raw$run<-as.factor(astro.raw$run)
K<-5; folds.rf <- sample(1:K, n, replace = T); mtry<-c(2,3,4,5,6,13,14,15)
CV.rf<-rep(1,15)
number.test<-matrix(nrow=5000,ncol = 1)
for (index in 1:5000){
  number.test[index,1]<-index}
for (i in mtry){
  CV.rf[i]<-0;
  for (j in 1:K){
    number.cv.train.rf<-number.test[folds.rf != j,]
    number.cv.test.rf<-number.test[folds.rf == j,]
    rf.astro<-randomForest(as.factor(class)~.,data=astro.raw
                           ,ntree=150,subset=number.cv.train.rf,mtry=i)
    yhat.rf<-predict(rf.astro,newdata=astro.raw[number.cv.test.rf,],type='class')
    perf.rf <-table(astro[number.cv.test.rf,]$class,yhat.rf)
    CV.rf[i]<-CV.rf[i]+1-sum(diag(perf.rf))/(n/5)
    }
  CV.rf[i]<-CV.rf[i]/5
}
error.rf<-CV.rf[which.min(CV.rf)]
error.rf
```

```
## [1] 0.0096
```

```r
which.min(CV.rf)
```

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

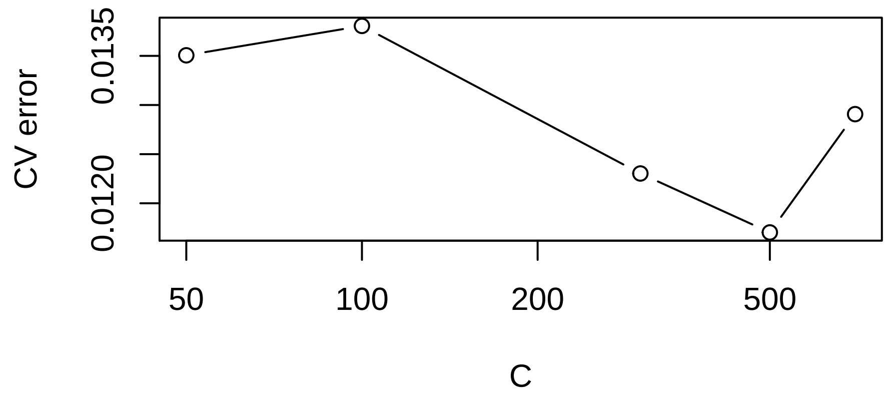
SVM

The biggest advantage of SVM is that it produces nonlinear boundaries by constructing a linear boundary in a large, transformed version of the feature space. We can use kernel function to generate these. So we will try to test three kernel functions: polynomial, Gaussian and Hyperbolic Tangent Kernel(MLP kernel) to see which is the best.

```
##                 rbfdot             polydot           tanhdot
## 1 0.0461630695443646 0.0143884892086331 0.436450839328537
```

Polynomial kernel is the best. So we choose "polydot" kernel fonction to do the cross validation to decide the parameter C. C is a hyperparameter, which balances training error and model complexity. It determines the margin of decision boudary. The margin is smaller for larger C. Hence larger values of C focus attention more on points near the decision boundary, while smaller values involve data further away. A large value of C will lead to an overfit boundary in the original feature space. We need to find the suitable C. In cross validation, we need to be careful that the observation which isn't a support vector won't effect the performance of the SVM, because the hyperplan of SVM is only created by support vectors.

```
CC<-c(50,100,300,500,700)
N<-length(CC)
err<-rep(0,N)
for(i in 1:N){
  err[i]<-cross(ksvm(as.factor(class)~.,data=data_train.raw,type="C-svc",
                     kernel="polydot",C=CC[i],cross=5, kpar="automatic"))
}
CV.SVM<-CC[which.min(err)]
plot(CC,err,type="b",log="x",xlab="C",ylab="CV error")
```



We will choose the CV accroding to the figure. And the test error is as follow:

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

```
## [1] 0.01438849
```

To summarise, if we consider the prediction error of each model overall, RandomForest is the best model with the smallest predection error and the most stable for the problem of exercice I.

```
##          LDA        QDA Multinomial_logistic_regression Naive_Bayes
## 1 0.08333333 0.01678657                       0.1918465   0.0263789
##          KNN     RF        SVM
## 1 0.1846523 0.0096 0.01438849
```

Part II Rendement du maïs

The purpose of Exercise II is to train a model to predict the yield of corn accroding to the data of weather. The variable to predict "yield_anomaly" is continuous which leads us to a regression problem. We start with loading the file CSV of data "mais_train".

We have a quick look of our data. Firstly, we find the predictor "X" which isn't presented in the document. So we delete it. Then we find out that there are some qualitive predictors which are "year_harvest", "NUMD", "IRR". These predictors are not climate data but influenced by the climate. So we still want to keep them. We transform then into dummy variables. Before transforming them, we know that we have 57 different "year_harvest" numbers, 94 "NUMD" numbers and 5 "IRR" numbers. There are too many classes for "year_harvest", "NUMD". So we decide to put them in several intervals. After checking the distribution of the data "year_harvest", "NUMD", we decide to cut "year_harvest" into 5 intervals, and "NUMD" into 5, too. Finally, the rest of the predictors are quantative. We normalize them for improving the chance and speed of convergence in fitting data.

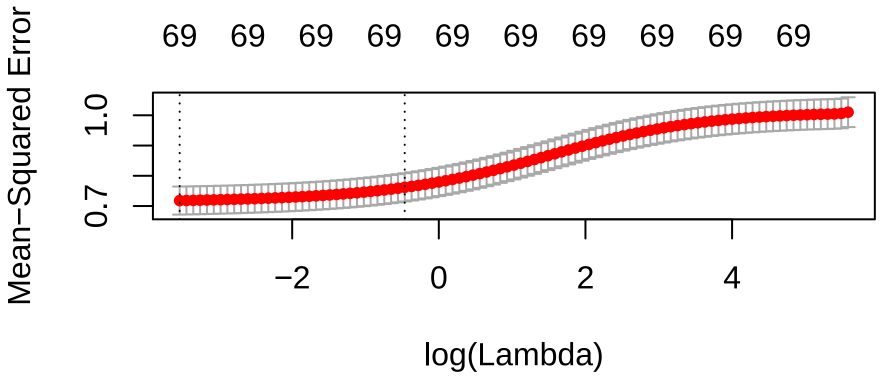
```r
mais<-mais.raw;
mais$year_harvest<-cut(mais$year_harvest, c(0,20,30,40,50,60))
mais$NUMD<-cut(mais$NUMD,c(0,20,40,60,80,100))
mais<-dummy_cols(mais, select_columns = "year_harvest");
mais<-mais[,-which(names(mais)=='year_harvest')];
mais<-dummy_cols(mais, select_columns = "NUMD");
mais<-mais[,-which(names(mais)=='NUMD')];
mais<-dummy_cols(mais, select_columns = "IRR");
mais<-mais[,-which(names(mais)=='IRR')];
mais[,2:55]<- scale(mais[,2:55])
```

We've confirmed that there is no missing data. And divide the data into training data and testing data.

Obviously, we can't fit model directly with our data because we have too many predictors, and there are a lot of 0 in our data because of dummy variables. So we will try to use shrinkage methods. Firstly, we try to add the penalty on the number of predictors by using Ridge, Lasso and Elastic Net.

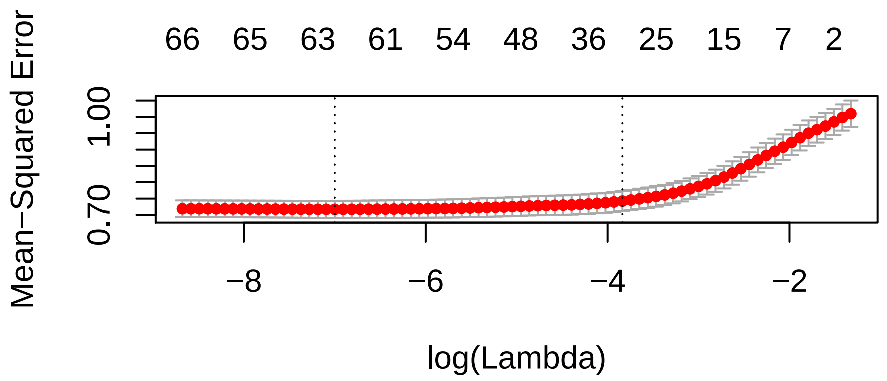
Ridge, Lasso and Elastic Net

We use cross-validation to find the best complexity parameter "lambda". The bigger the lambda is ,the more penalty is added on the number of predictors. And in the case of elastic net, we also use the cross validation to choose alpha to find the best



1.Ridge

In the figure above, although it tells us the minimum mean-squared error is showed up in the figure , it's not clear enough. Ridge has one obvious disadvantage, it will include all the predictors in the final model because it use L2 penalty. We think this may cause the bad performance. So we try to use Lasso.



2.Lasso

In the figure above, it's better than the last one. We can find the minimum mean-squared error is appeared in the figure. Lasso use L1 penalty,and it has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter lambda is sufficiently large. It will remove some predictors, and that is what we want.

3.Elastic Net

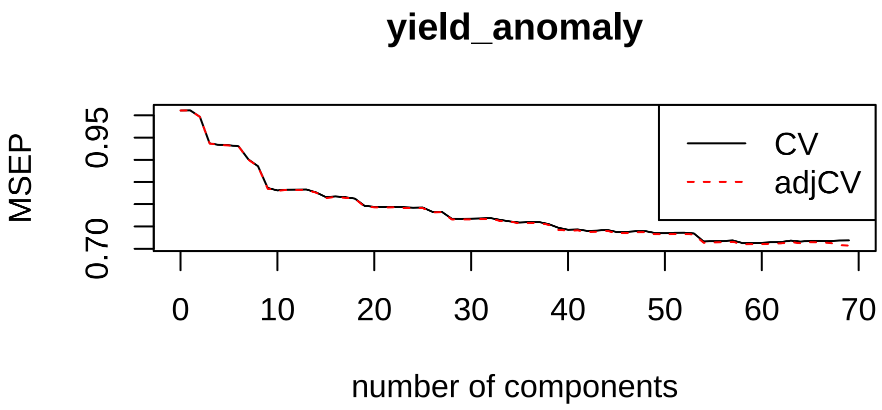
However, Lasso often tends to "over-regularize" a model that might be overly compact and therefore under-predictive.The Elastic Net is used to fix this problem by balancing between Lasso and Ridge penalties. We also use corss validation to choose the best alpha.

The result of elastic-net is very simliar to the Lasso. We showed the mean-squared error of three algorithmes as below.

```
##       Ridge     Lasso   Elastic
## 1 0.6555697 0.6409263 0.6489272
```

Principal component regression (PCR)

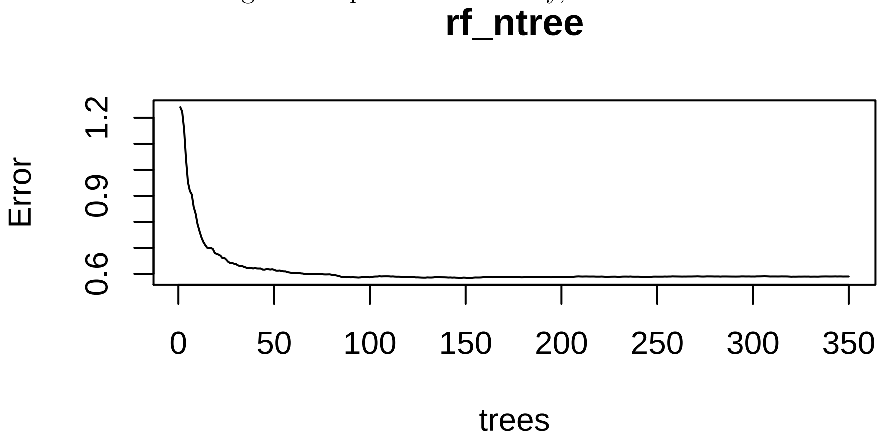
Now we will try to use "Feature extraction". Different from the methods of Rridge and lasso which just reduce the number of the predictors, "Feature extraction" construct new predictors from the initial predictors. The idea is to reduce the input dimension (and, consequently, the number of parameters), while keeping most of information relevant to predict. Firstly, we try to use a unsupervised method, Principal Component Analysis, to create new predictors and use them in our regression. It's call Principal component regression (PCR). The number of new predictors can be finded by cross-validation.



In this figure, we can know the mean-squared error keeps going down when the number of predictors increase. It finally create 69 predictors. It doesn't reduce the number of predictors. That explian why we have the high mean-squared error.

Regression trees

We have used decision tree to solve the classification problem in Ex1, and we have analyzed the advantages and the disadvantages of decision trees. Here, we also use bagging and randomForest to solve our regression problem. Firstly, we need to decide how many trees we want to make.



It seems 200 is not bad. Now, we make 200 trees. To use bagging, we set mtry as the number of the predictors before having dummy variables. Because Tree don't need to use dummy variables. To use randomForest, we set mtry as sqrt(number of the predictors).

```r
K<-5; folds.rf <- sample(1:K, n, replace = T); mtry<-c(6,7,8,56,57)
CV.rf<-rep(1,57)
number.test<-matrix(nrow=2300,ncol = 1)
for (index in 1:2300){
  number.test[index,1]<-index}
for (i in mtry){
  CV.rf[i]<-0
  for (j in 1:K){
    number.cv.train.rf<-number.test[folds.rf != j,]
    number.cv.test.rf<-number.test[folds.rf == j,]
    rf<-randomForest(yield_anomaly~.,data=mais.raw,ntree=200
                     ,subset=number.cv.train.rf,mtry=i)
    yhat<-predict(rf,newdata=mais.raw[number.cv.test.rf,])
    CV.rf[i]<-CV.rf[i]+mean((mais.raw$yield_anomaly[number.cv.test.rf] - yhat)^2)}
  CV.rf[i]<-CV.rf[i]/5}
error.rf<-CV.rf[which.min(CV.rf)]
error.rf
```

```
## [1] 0.5668675
```

The result is very good.

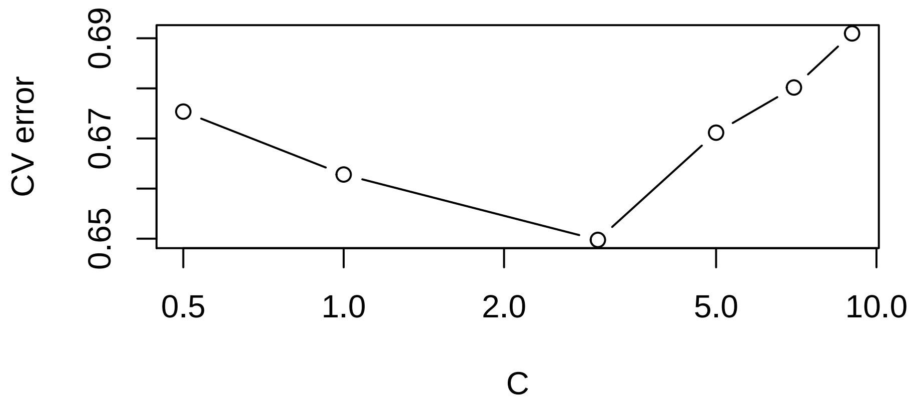
Support Vector Regression(SVR)

SVMs were first developed for classification. We have used and analyzed them in the Ex1. To generalize the SV algorithm to regression, we need to find a way of retaining this feature. This can be achieved using the epsilon-insensitive loss function. If |f (x) - y| <= epsilon,then f (x) - y = 0 where f (x) is predicted value and y is true value. Epsilon specifies the desired accuracy of the approximation. Now, we need to use cross validation to find the best epsilon and C, and C is a hyperparameter, which balances training error and model complexity. Before tuning them, we need to choose to a right kernel function.

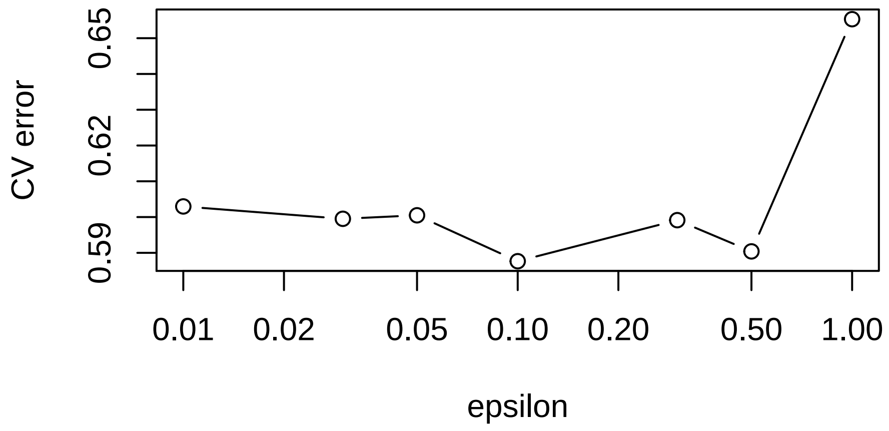
```
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
##              rbfdot          polydot         tanhdot
## 1 0.564447301411097 0.70609571425012 15812.752919342
```

Gaussian kernel is the best. Now, we start with tuning C.



We will choose the C accroding to the figure above. Then we turn to choose epsilon.



We can choose the epsilon accroding to the figure above. And we will use these two parameters to predict our test error as below.

```
## [1] 0.5483959
```

To summarize, if we consider the prediction error of each model overall, SVR is the best model with the smallest predection error for the problem of exercice II.

```
##       Ridge     Lasso Elastic_net       PCR Regression_tree       SVR
## 1 0.6555697 0.6409263   0.6489272 0.7289778       0.5668675 0.5483959
```

Part III Images naturelles

We are going to use Convolution Neural Network(CNN) to create our classifier. CNN is presented in the class, and it's good at classifying the images. That's the reason we choose it.

0. Preparations

0.1 Reading images

We have 485 images of the car, 590 images of the cat and 521 images of the flower. We use EBImage to read images and put images into 3 List, photo.car, photo.cat and photo.flower. It's not the interesting part we want to present, so we hide the codes for saving the place.

0.2 Resizing images

By using the function str(), we can have a quick look of all the images. We notice that the sizes of the images are not the same. It's important to resize them for putting the images into input layer. We decide to resize all the images into 32*32 with 3 channels which are green, red, blue.

```r
resizeImage<-function(car, cat, flower, size){
  photo.car<-list(); photo.cat<-list(); photo.flower<-list()
  for(i in 1:485){photo.car[[i]]<-resize(car[[i]],size,size)}
  for(i in 1:590){photo.cat[[i]]<-resize(cat[[i]],size,size)}
  for(i in 1:521){photo.flower[[i]]<-resize(flower[[i]],size,size)}
  images<-list("car"=photo.car, "cat"=photo.cat, "flower"=photo.flower)
  return(images)
}
```

0.3 Transforming data

Now we have 3 lists of image, we need to transform them into the input accepted by the functions of Keras.Firstly, we combine them into one list.Then, we split all data into training set and testing set. Finally, we transform the input into an array with 4 dimensions(number_of_image, height, width, number_of_channel) which fits for the input layer of CNN created by Keras. Besides, we are going to create labels to present the objects in images. We decide to use 0 for car, 1 for cat and 2 for flower, and we use one-hot-encoding to transform the labels to fit outputs of CNN.

```r
transformData<-function(photo.car, photo.cat, photo.flower, n){
  # Combine data
  dataX<-list();
  for(i in 1:n){dataX[[i]]<-photo.car[[i]]}
  for(i in (n+1):(2*n)){dataX[[i]]<-photo.cat[[i-n]]}
  for(i in (2*n+1):(3*n)){dataX[[i]]<-photo.flower[[i-2*n]]}
  y0<-rep(0,n); y1<-rep(1,n); y2<-rep(2,n)
  dataY<-c(y0,y1,y2)
  #split data into training set and testing set
  nb<-rep(1:(3*n),1)
  ntrain<-sample(1:(3*n),size = (2*n), replace = FALSE)
  ntest<-nb[-ntrain]
  trainY<-dataY[ntrain]; testY<-dataY[-ntrain]
  trainX<-list(); testX<-list()
  j<-1
  for(i in ntrain){trainX[[j]]<-dataX[[i]]; j<-j+1}
  j<-1
  for(i in ntest){testX[[j]]<-dataX[[i]]; j<-j+1}
  # transform data for the input layer of CNN
  # create input features
  test.x<-combine(testX)
  train.x<-combine(trainX)
  # reorder the dimension
  test.x<-aperm(test.x, c(4,1,2,3))
  train.x<-aperm(train.x, c(4,1,2,3))
  # create labels
  train.labels<-to_categorical(trainY)
  test.labels<-to_categorical(testY)
  newData<-list("trainX"=train.x, "testX"=test.x
               , "trainLabels"=train.labels, "testLabels"=test.labels, "testY"=testY)
  return(newData)
}
```

1. Modeling

We use "Complex Layer technology" given in the class. Firstly, we create a new layer which has 2 convolution layers and one max pooling layer. Convolution layer is used for detecting the boundaries of the object in the image. We use 3*3 as the filter. All the filters in the same convolution layer share the same parameters, and the input matrix of the image will lose 2 lines and columns after getting through one convolution layer because the filter is 3*3. In the convolution layer, we use "relu" as the activation function. After passing through 2 convolution layer, the input gets into max pooling layer, and the heigth and the width will be half. That's why we add 2 convolution layers. It can make sure the number of lines or columns is even after being half.

We use the layer that we created twice. In the first time, the number of filter is 32, and the second is 64. The more filters we add, the more information about the boudaries we have, which helps us detect the object.

Now, the input becomes smaller(height and width are reduced by max pooling layer significantly), but much deeper than before(the depth is increase by the convolution layer). We flatten the input into one dimsension, and use 2 fully connected layers to calculate the weigth of each unit, and the activation function of the fully connected layer is "relu".To prevent overfitting, we use dropout layer with the rate = 0.25 after each fully connected layer. In the end, we use "softmax" function to calculte the probabilities of the input being car, cat or flower.

```
model<-keras_model_sequential()
model %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu'
               , input_shape = c(32,32,3)) %>%
  layer_conv_2d(filters = 32, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_conv_2d(filters = 64, kernel_size = c(3,3), activation = 'relu') %>%
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  layer_flatten() %>%

  layer_dense(units = 1024, activation = 'relu') %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 256, activation = 'relu') %>%
  layer_dropout(rate = 0.25) %>%
  layer_dense(units = 3, activation = 'softmax') %>%
compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_sgd(lr=0.01, momentum = 0.9),
  metrics = c('accuracy'))
summary(model)
```

```
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## conv2d_1 (Conv2D)                (None, 30, 30, 32)            896
## ___________________________________________________________________________
## conv2d_2 (Conv2D)                (None, 28, 28, 32)            9248
## ___________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)   (None, 14, 14, 32)            0
## ___________________________________________________________________________
## conv2d_3 (Conv2D)                (None, 12, 12, 64)            18496
## ___________________________________________________________________________
## conv2d_4 (Conv2D)                (None, 10, 10, 64)            36928
## ___________________________________________________________________________
## max_pooling2d_2 (MaxPooling2D)   (None, 5, 5, 64)              0
## ___________________________________________________________________________
## flatten_1 (Flatten)              (None, 1600)                  0
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 1024)                  1639424
## ___________________________________________________________________________
## dropout_1 (Dropout)              (None, 1024)                  0
## ___________________________________________________________________________
## dense_2 (Dense)                  (None, 256)                   262400
## ___________________________________________________________________________
## dropout_2 (Dropout)              (None, 256)                   0
## ___________________________________________________________________________
## dense_3 (Dense)                  (None, 3)                     771
## ===========================================================================
## Total params: 1,968,163
## Trainable params: 1,968,163
## Non-trainable params: 0
## ___________________________________________________________________________
```

2. Fiting model

The loss function for classification is "cross-entropy", so we use it. For the optimizer, we use "stochastic gradient descent". We set the learning rate as 0.01, and set momentum = 0.9 to accelerate learning.

```
images<-resizeImage(car = car, cat = cat, flower = flower, size = 32)
data<-transformData(photo.car = images$car, photo.cat = images$cat
                    , photo.flower = images$flower, n = 250)
fit<-model %>% fit(data$trainX, data$trainLabels, epochs = 50, batch_size = 32,
                   validation_data = list(data$testX, data$testLabels))
```

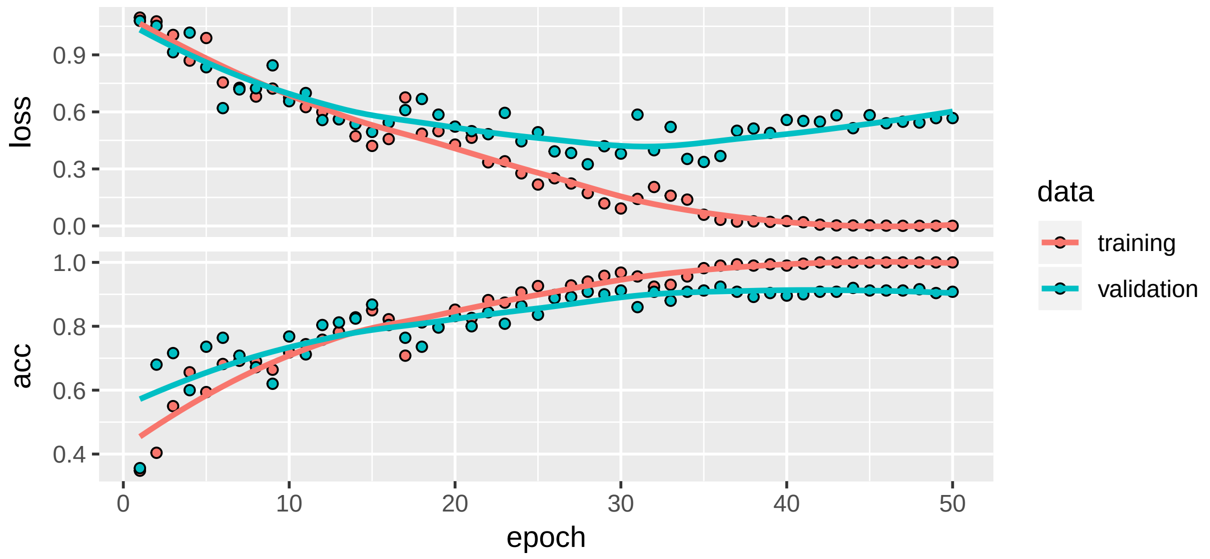
3. Evaluation

The loss and accuracy of classification on testing set are as below.

```
model %>% evaluate(data$testX, data$testLabels)
```

```
## $loss
## [1] 0.5673323
##
## $acc
## [1] 0.908
```

```
plot(fit)
```



In the figure of loss. we can see the loss in training set keeps going down, and it is close to 0 in training set, which is good. But in testing set, it doesn't, which is not good. In the figure of accuracy, we can see that 2 curves are close which means we avoid overfitting. In the confusion matrix below, we can see that the model works not bad.

```
pred<-model %>% predict_classes(data$testX)
table(Predicted = pred, Actual = data$testY)
```

```
##          Actual
## Predicted  0  1  2
##         0 74  2  4
##         1  4 78  7
##         2  4  2 75
```

4. The ways to improve the performance

There are some methods to improve the performance. Firstly, we can resize images bigger. Now the images are 32 * 32. We can make them 300 * 300, which can show more details of the boundaries, and helps classify the images. But it costs more space and time to fitting the model. Or, we can increase the number of the image in training set. Now we have 500 images in training. But we can use all 1600 images to train the model. And it also costs more space and time. What's more, we can add more convolution layers and add more filters in them which helps get more information about the boundaries.