Feature selection

(Henry T.H. Tu, 26-sep-2015)

# 1. Image to vector

### 1.1. The motivation to turn an image into a vector

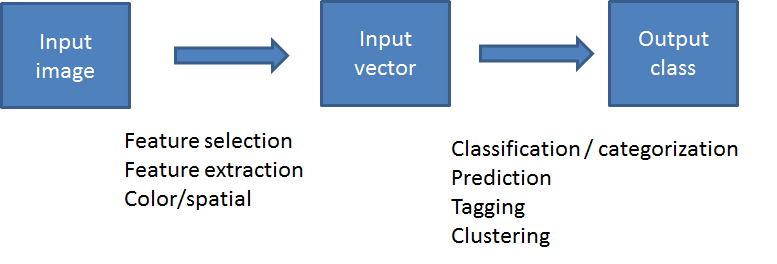
+ Learning methods (svm, linear model, logistic model, decision tree) only work on vectors. But we are given images, documents.

+ We need to convert documents/images into vector. This is feature extraction/selection

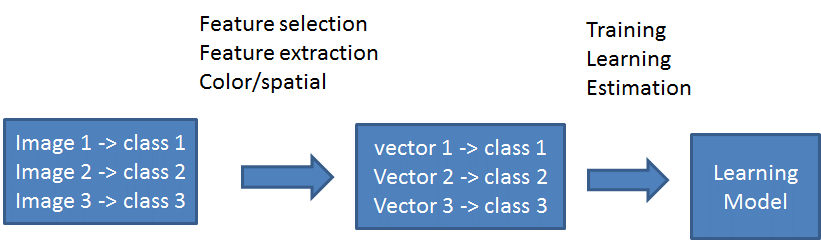
+ Feature selection is not usually a part in machine learning books. However, it can be found in natural language processing, computer vision, image processing books.

### 1.2. The real learning pipeline

+ input image >> input vector >> output class



+ input images >> input vectors >> learning model/error



### 1.3. Representing images in R

+ each image is an array of 3 dimensions, img[1:row, 1:col, 1:3]

+ we can separate the image into 3 channels (red, green, blue)

|  |
| --- |
| library(jpeg);  img <- readJPEG("C:/opt/images/mit8-images/coast\_arnat59.jpg");  print( img[1:5, 1:7, ] );  r\_ch <- img[, , 1]; #first channel  g\_ch <- img[, , 2]; #second channel  b\_ch <- img[, , 3]; #third channel  print( dim(r\_ch) );  print( dim(g\_ch) );  print( dim(b\_ch) ); |

+ And example of image

|  |  |
| --- | --- |
| , , 1  [,1] [,2] [,3] [,4] [,5] [,6] [,7]  [1,] 0.2470588 0.2509804 0.2470588 0.2470588 0.2431373 0.2392157 0.2431373  [2,] 0.2470588 0.2470588 0.2549020 0.2509804 0.2470588 0.2549020 0.2627451  [3,] 0.2431373 0.2431373 0.2509804 0.2509804 0.2588235 0.2627451 0.2549020  [4,] 0.2509804 0.2549020 0.2627451 0.2509804 0.2549020 0.2627451 0.2549020  [5,] 0.2549020 0.2588235 0.2588235 0.2588235 0.2627451 0.2627451 0.2509804  , , 2  [,1] [,2] [,3] [,4] [,5] [,6] [,7]  [1,] 0.3450980 0.3490196 0.3450980 0.3450980 0.3411765 0.3372549 0.3411765  [2,] 0.3450980 0.3450980 0.3529412 0.3490196 0.3450980 0.3529412 0.3607843  [3,] 0.3529412 0.3411765 0.3490196 0.3529412 0.3607843 0.3647059 0.3568627  [4,] 0.3490196 0.3529412 0.3607843 0.3490196 0.3568627 0.3647059 0.3568627  [5,] 0.3529412 0.3568627 0.3490196 0.3490196 0.3607843 0.3607843 0.3490196  , , 3  [,1] [,2] [,3] [,4] [,5] [,6] [,7]  [1,] 0.5019608 0.5058824 0.5019608 0.5019608 0.5019608 0.4980392 0.5019608  [2,] 0.5019608 0.5019608 0.5098039 0.5058824 0.5019608 0.5098039 0.5176471  [3,] 0.5098039 0.4980392 0.5058824 0.4980392 0.5058824 0.5098039 0.5019608  [4,] 0.5098039 0.5137255 0.5176471 0.5058824 0.5019608 0.5098039 0.5019608  [5,] 0.5098039 0.5137255 0.5098039 0.5098039 0.5176471 0.5176471 0.5058824 | + 3 channels of the image (red, green, blue)  + each channel has the same size (width, height) |

# 2. Color histogram

### 2.1. Motivation

+ the count of the red values, green values, blue values can be the feature for image?

### 2.2. Method

+ if we count all colors in the image, then the space of color can be very huge 256\*256\*256 dimension.

|  |  |
| --- | --- |
|  | Color histogram.  Just take all the color values in the image and count the color.  This will be huge when we have 256\*256\*256 colors. We have to shrink the histogram |

+ We can compute the scaled color. (r, g, b) = (r/16, g/16, b/16). Therefore we only have the values from (0, 0, 0) to (15, 15, 15) which is (4096 entries).

+ Many similar colors (1, 0, 0) and (2, 0, 0) will be put into the same bin in counting.

+ Many similar colors (15, 0, 123) and (16, 0, 123) may be in the different bins. Although they are very close to each other.

|  |  |
| --- | --- |
|  | Color histogram 16\*16\*16  We scale the color to lower range by dividing with 16.  (r, g, b) = (r/16, g/16, b/16)  We count the values within the range (0, 0, 0) to (16, 16, 16) which is smaller. |

### 2.3. R code for color histogram

|  |
| --- |
| rm(list = ls());  library(jpeg);  colorHist <- function(img)  {  img <- floor( floor(255 \* img )/ 16 );  mr = nrow(img[,,1]);  mc = ncol(img[,,1]);  h <- array(0\*1:4096, c(16, 16, 16) );  for(r in 1:mr) for(c in 1:mc)  {  rk <- img[r, c, 1] + 1;  gk <- img[r, c, 2] + 1;  bk <- img[r, c, 3] + 1;  h[rk, gk, bk] <- h[rk, gk, bk] + 1;  }  return( as.integer(h) );  }  # to read all the files in one folder  files <- list.files("C:/Users/henrytu/Desktop/images", full.name=TRUE);  i1 <- readJPEG(files[1]); h1 <- colorHist(i1); print( c(min(h1), max(h1), which(h1>0) ));  i1 <- readJPEG(files[2]); h1 <- colorHist(i1); print( c(min(h1), max(h1), which(h1>0) ) );  i1 <- readJPEG(files[3]); h1 <- colorHist(i1); print( c(min(h1), max(h1), which(h1>0) ) ); |

### 2.4. R code for color histogram table

|  |
| --- |
| rm(list = ls());  library(jpeg);  colorHist <- function(img)  {  img <- floor( floor(255 \* img )/ 16 );  mr = nrow(img[,,1]);  mc = ncol(img[,,1]);  h <- array(0\*1:4096, c(16, 16, 16) );  for(r in 1:mr) for(c in 1:mc)  {  rk <- img[r, c, 1] + 1;  gk <- img[r, c, 2] + 1;  bk <- img[r, c, 3] + 1;  h[rk, gk, bk] <- h[rk, gk, bk] + 1;  }  return( as.integer(h) );  }  files <- list.files("C:/opt/images/mit8-images", full.name=TRUE);  files <- files[1:20];  colorHistTable <- function(files, dpar = 4096)  {  mr <- length(files);  df <- matrix(nrow=mr, ncol=dpar);  for(k in 1:mr) df[k, ] <- colorHist( readJPEG(files[k]) );  return(df);  }  X <- colorHistTable(files);  X[, 1:7] |

# 3. Color averaging

### 3.1. Motivation

+ can we summarize an image with small vectors?

+ the simplest method would be color averaging

### 3.2. Method

r = average all values in red channels

g = average all values in green channels

b = average all values in blue channels

v = (r, g, b) is the feature vector with length 3, which is very small.

### 3.3. R code for color averaging

+ Please notice that we don't load all the files, which can be very slow. But we will load some first files for testing

|  |  |
| --- | --- |
| rm(list = ls());  library(jpeg);  colorAvg <- function(img)  {  r <- mean(img[,,1]);  g <- mean(img[,,2]);  b <- mean(img[,,3]);  return( c(r, g, b) );  }  colorAvgTable <- function(files)  {  X <- matrix(nrow=length(files), ncol=3);  for(k in 1:length(files))  {  ik <- readJPEG(files[k]);  X[k, ] <- colorAvg(ik);  }  return(X);  }  files <- list.files("C:/opt/images/mit8-images", full.name=TRUE);  X <- colorAvgTable(files[1:10]);  X |  |

# 4. Spatial color averaging

### 4.1. Motivation

+ The motivation is to differentiate the sky/grass and grass/sky. They will be the same when we use color averaging or color histogram.

+ We introduce the spatial pyramid (Lazebnik 2006) to gain more spatial information.

### 4.2. Method

|  |  |
| --- | --- |
|  |  |

### 4.3. R coding for spatial color averaging

|  |  |
| --- | --- |
| rm(list = ls()); library(jpeg);  colorAvg <- function(img)  {  r <- mean(img[,,1]);  g <- mean(img[,,2]);  b <- mean(img[,,3]);  return( c(r, g, b) );  } | loading layer 1 |
| colorAvg2 <- function(img)  {  mr = nrow(img);  mc = ncol(img);  mr <- mr / 2;  mc <- mc / 2;  a1 <- img[1:mr, 1:mc, ];  a2 <- img[1:mr + mr, 1:mc, ];  a3 <- img[1:mr, 1:mc + mc, ];  a4 <- img[1:mr + mr, 1:mc + mc, ];    v1 <- colorAvg(a1);  v2 <- colorAvg(a2);  v3 <- colorAvg(a3);  v4 <- colorAvg(a4);    m <- rbind(v1, v2, v3, v4);  return(m);  } | loading layer 2 |
| colorAvg3 <- function(img)  {  mr = nrow(img);  mc = ncol(img);  mr <- mr / 2;  mc <- mc / 2;  a1 <- img[1:mr, 1:mc, ];  a2 <- img[1:mr + mr, 1:mc, ];  a3 <- img[1:mr, 1:mc + mc, ];  a4 <- img[1:mr + mr, 1:mc + mc, ];    v1 <- colorAvg2(a1);  v2 <- colorAvg2(a2);  v3 <- colorAvg2(a3);  v4 <- colorAvg2(a4);    m <- rbind(v1, v2, v3, v4);  return(m);  } | loading layer 3 |
| colorAvg123 <- function(img)  {  m1 <- colorAvg(img);  m2 <- colorAvg2(img);  m3 <- colorAvg3(img);  m <- rbind(m1, m2, m3);  return( as.vector(m) );  } | loading 3 layers |
| colorAvgSpatialTable <- function(files, dpar=21\*3)  {  X <- matrix(nrow=length(files), ncol=dpar);  for(k in 1:length(files))  {  ik <- readJPEG(files[k]);  X[k, ] <- colorAvg123(ik);  }  return(X);  } | loading table |
| files <- list.files("C:/opt/images/mit8-images", full.name=TRUE);  X <- colorAvgSpatialTable(files[1:10]);  X[, 1:7] | Testing the function |

# 5. Spatial color histogram

### 5.1. The motivation

+ Same as color averaging, can we differentiate sky/grass and grass/sky?

### 5.2. Method

+ Similar to color averaging, we can apply color histogram on cells to obtain the spatial features.

+ There will be 21\*4096 matrix for each input image when we use 3 layers (16+4+1) and color histogram of 3 channels (16\*16\*16)