Lecture notes, week 1

***(Henry T.H. Tu, 21-sep-2015)***

A1. logistic regression: binomial case 2

A1.1. The Facebook motivation to logistic regression 2

A1.2. The logistic model for binary classification 2

A1.3. Learning with Gradient Descending 3

A1.4. RMD code 4

A2. logistic regression: multinomial case 4

A2.1. Motivation 5

A2.2. The extended logistic regression for multinomial classification 5

A2.3. RMD code 6

A3. Learning concepts 7

A3.1. Under- and over-fitting 7

A3.2. The model selection pipeline 8

A3.3. The procedure 8

A3.4. RMD code 9

A3.5. Linearly separable 10

A3.6. Linearly non-separable 11

B1. Mixture models 12

B1.1. p = 0.3\*Gaussian + 0.7\*Gaussian 12

B1.2. RMD: to show mixture 13

B1.3. p = 0.3\*Uniform + 0.7\*Uniform 14

B1.4. RMD: to mix or to combine uniform distributions 14

B1.5. The mixture model 15

B1.6. Generalized weights 16

B2. The k-means procedure 16

B2.1. The compression motivation 17

B2.2. We can measure the errors (differences) between the vectors 17

B2.3. Comparing two vocabularies 18

B2.4. The k-mean procedure 19

B2.5. RMD code 19

B3. The E-M procedure (expectation maximization) 21

B3.1. Extension to many labels 21

B3.2. The E-step / labeling step 21

B3.3. The M-step / averaging step 22

B3.4. The whole procedure 22

B4. Gaussian mixture model 22

B4.1. Motivation 23

B4.2. The Gaussian mixture model 23

B4.3. Learning GMM with EM method 23

# A1. logistic regression: binomial case

### A1.1. The Facebook motivation to logistic regression

|  |  |
| --- | --- |
|  | Figure: the toy face detection example.  We have a picture with many faces, can we show rectangle around the faces?  Can we divide the faces into 3 classes (up, down, middle) based on the shape of the faces?  This is similar to the facebook face tagging function. We have to work with real data in facebook case. In our case, the toy data is simple enough to illustrate the idea and concepts. |

+ Note that this is not what happens in the real pictures. But it does show the same process of face detection.

+ We have to location the rectangles around the faces (by segmentation or by feature point detection). Then we have to apply the logistic function (discriminant function) on the location to see if it has strong response to the face class.

### A1.2. The logistic model for binary classification

+ We model the faces with and background



+ The decision rule for each input x



+ In many programming languages, we can write

y = ( p(face, x) > p(back, x) ? "face" : "back" );

### A1.3. Learning with Gradient Descending

+ Given the dataset D = { x1->y1, x2->y2, .., xk->yk }.

+ We need to fit the logistic model  to the dataset D.

+ All we have to do is to learn w from dataset D.

+ We can base on the cost function



+ The gradient descending procedure to start from w0 = (0, 0, 0, .. 0) and to arrive at the optimal point w\* after several steps.

|  |  |
| --- | --- |
|  | Figure: the linear dataset |
|  | The linear coefficients corresponding to the dataset |

### A1.4. RMD code

|  |
| --- |
| # To show how logistic regression works  ```{r}  rm(list=ls()); library(nnet);  ```  ## 1. Creating the linear dataset  ```{r}  x1 <- runif(5000, -10, 10);  x2 <- runif(5000, -20, 20);  y <- (2\*x1 + 3\*x2 > 0);  y <- as.factor(y);  df <- data.frame(x1, x2, y);  print( qplot(x1, x2, color=y) );  print( head(df) );  ```  ## 2. Training the logistic model  ```{r}  mod <- multinom(y ~ x1 + x2, df);  print( mod );  ```  ## 3. Test the model on the training data  ```{r}  y1 <- predict(mod, subset(df, select=-c(y) ) );  e1 <- abs( y != y1 );  print(head(data.frame( y, y1, e1 )));  print(data.frame( key="Training error", val=sum(e1) ));  print( table(y, y1) );  ``` |

# A2. logistic regression: multinomial case

### A2.1. Motivation

|  |  |
| --- | --- |
|  | We have the picture of many animals.  Can we give tag for each animal? |
|  |  |

### A2.2. The extended logistic regression for multinomial classification

+ The probability model: for each class (dog, cat, cow), we will have one probability function against the background

|  |  |
| --- | --- |
|  | Equation: the general logistic model |

+ The decision rule: we choose the class yk with strongest response p(yk|x) to be the output class.

|  |  |
| --- | --- |
|  | Equation: the decision rule for the general logistic model |

### A2.3. RMD code

|  |
| --- |
| # To show how logistic regression works  ```{r}  rm(list=ls()); library(nnet); set.seed(1234);  ```  ## 1. Creating the xor dataset  ```{r}  x1 <- runif(2000, -10, 10);  x2 <- runif(2000, -20, 20);  y <- 2\*(x1\*x2 > 0) + (x1 > 0);  y <- as.factor(y);  df <- data.frame(x1, x2, y);  print( qplot(x1, x2, color=y) );  print( head(df) );  ```  ## 2. Training the logistic model  ```{r}  mod <- multinom(y ~ x1 + x2, df);  print( mod );  ```  ## 3. Test the model on the training data  ```{r}  y1 <- predict(mod, subset(df, select=-c(y) ) );  e1 <- abs( y != y1 );  print(head(data.frame( y, y1, e1 )));  print(data.frame( key="Training error", val=sum(e1) ));  print( table(y, y1) );  ``` |

# A3. Learning concepts

### A3.1. Under- and over-fitting

|  |  |
| --- | --- |
|  | Figure: when we try to fit the linear model to the quadric dataset, we will have a lot of error (under-fitting) |
|  | Figure: on the other extreme, when we try to fit the high-degree polynomial to the quadric dataset, it becomes over-fitting.  The training error can be very small. But then the model cannot handle new input. It will produce large error with new input not in the dataset |
|  | Figure: so we want the model with small training error (fitting the dataset very well) and small testing error (being able to handle new input) |

### A3.2. The model selection pipeline

|  |
| --- |
|  |
| Figure: given a dataset D, we have to find the model M to fit in the dataset. We have to base on both training and testing error to select the model |

### A3.3. The procedure

|  |
| --- |
|  |
| Figure: We are given the dataset D, we have to find the model M to fit the data. We need to go through several steps to arrive at the model.  Remember that we care not only about the model M but also the error/confidence of the model M subject to the dataset D. |

### A3.4. RMD code

|  |
| --- |
| # To show the training/testing procedure  ```{r}  rm(list=ls()); library(caret); cat( rep("\n", 10) );  ```  ## 1. Loading data  ```{r}  x <- 1:7;  y <- c(5, 10, 14.5, 20, 25, 31, 35)  D <- data.frame(x, y); print( head(D) );  ```  ## 2. Splitting data  ```{r}  A <- D[1:4, ]; print( head(A) );  B <- D[5:6, ]; print( head(B) );  ```  ## 3. Training a linear model  ```{r}  M <- lm(y ~ x, A);  print( coef(M) );  ```  ## 4a. Testing if the model fits the data  ```{r}  hA = predict(M, A);  gA = A$y;  print(data.frame(gnd=gA, hat=hA, err=abs(gA-hA) ));  ```  ## 4b. Testing if the model can handle new input  ```{r}  hB = predict(M, B); gB = B$y;  print(data.frame(gnd=gB, hat=hB, err=abs(gB-hB) ));  ``` |

### A3.5. Linearly separable

|  |  |
| --- | --- |
|  | Figure: in some case, the dataset is easy to separate. A line is enough to divide the dataset into two classes (for example face and background). |
|  | Figure: we can simulate / generate the linearly separable dataset with R code  x <- runif(1000, -10, 10);  y <- runif(1000, -10, 10);  z <- ( 2\*x + 3\*y > 0);  print( qplot(x, y, color=z) ); |

### A3.6. Linearly non-separable

|  |  |
| --- | --- |
|  | Figure: in general, we will have to use complicate model to capture the data distribution.  For example, when the data is circular or elliptic, we need Gaussian distribution to model the data.  We can also use linear model on transformed feature, in this case, we transform x\*x and y\*y from x and y to model the circle. |
|  | Figure: we can simulate / generate the linearly separable dataset with R code  x <- runif(2000, -10, 10);  y <- runif(2000, -20, 20);  z <- ( 0.02\*x\*x + 0.03\*y\*y < 1);  print( qplot(x, y, color=z) ); |

# B1. Mixture models

### B1.1. p = 0.3\*Gaussian + 0.7\*Gaussian

|  |  |
| --- | --- |
|  |  |
| x <- -30:70;  m1 <- 5; s1 <- 5;  p1 <- exp(- (x-m1)\*(x-m1) / (2\*s1\*s1)) / sqrt( 2 \* pi \* s1 \* s1 );  plot(x, p1, type="l"); | m2 <- 30; s2 <- 5;  p2 <- exp(- (x-m2)\*(x-m2) / (2\*s2\*s2)) / sqrt( 2 \* pi \* s2 \* s2 );  plot(x, p2, type="l"); |

+ When we mix two of them

|  |  |  |
| --- | --- | --- |
|  |  |  |
| p <- 0.5\*p1 + 0.5\*p2;  plot(x, p, type="l"); | p <- 0.7\*p1 + 0.3\*p2;  plot(x, p, type="l"); | p <- 0.1\*p1 + 0.9\*p2;  plot(x, p, type="l"); |

+ If you are not familiar with Gaussian distribution, this is the formula



### B1.2. RMD: to show mixture

|  |
| --- |
| # To show how to mix two probability models  ## Preparation  ```{r}  rm(list=ls());  library(nnet); library(ggplot2);  set.seed(1234);  ```  ## 1. We generate gausian components  ```{r}  x <- -30:70;  m1 <- 5; s1 <- 5;  p1 <- exp(- (x-m1)\*(x-m1) / (2\*s1\*s1)) / sqrt( 2 \* pi \* s1 \* s1 );  plot(x, p1, type="l");  m2 <- 30; s2 <- 5;  p2 <- exp(- (x-m2)\*(x-m2) / (2\*s2\*s2)) / sqrt( 2 \* pi \* s2 \* s2 );  plot(x, p2, type="l");  ```  ## 2b. We mix them with equal weights  ```{r}  p <- 0.5\*p1 + 0.5\*p2;  plot(x, p, type="l");  ```  ## 2c. We mix them with equal weights  ```{r}  p <- 0.7\*p1 + 0.3\*p2;  plot(x, p, type="l");  ```  ## 2d. We mix them with equal weights  ```{r}  p <- 0.1\*p1 + 0.9\*p2;  plot(x, p, type="l");  ``` |

### B1.3. p = 0.3\*Uniform + 0.7\*Uniform

|  |  |
| --- | --- |
|  |  |
| x <- 1:50;  a <- 3; b <- 15; p1 <- (x>=a)\*(x<=b)/(b-a) + 0\*x;  plot(x, p1, type="l"); | a <- 18; b <- 30; p2 <- (x>=a)\*(x<=b)/(b-a) + 0\*x;  plot(x, p2, type="l"); |

+ We have the result

|  |  |  |
| --- | --- | --- |
|  |  |  |
| p <- 0.5\*p1 + 0.5\*p2;  plot(x, p, type="l"); | p <- 0.7\*p1 + 0.3\*p2;  plot(x, p, type="l"); | p <- 0.1\*p1 + 0.9\*p2;  plot(x, p, type="l"); |

### B1.4. RMD: to mix or to combine uniform distributions

|  |
| --- |
| # To show how to mix two probability models  ## Preparation  ```{r}  rm(list=ls());  library(nnet); library(ggplot2);  set.seed(1234);  ```  ## 1. We generate gausian components  ```{r}  x <- 1:50;  a <- 3; b <- 15; p1 <- (x>=a)\*(x<=b)/(b-a) + 0\*x;  plot(x, p1, type="l");  a <- 18; b <- 30; p2 <- (x>=a)\*(x<=b)/(b-a) + 0\*x;  plot(x, p2, type="l");  ```  ## 2b. We mix them with equal weights  ```{r}  p <- 0.5\*p1 + 0.5\*p2;  plot(x, p, type="l");  ```  ## 2c. We mix them with equal weights  ```{r}  p <- 0.7\*p1 + 0.3\*p2;  plot(x, p, type="l");  ```  ## 2d. We mix them with equal weights  ```{r}  p <- 0.1\*p1 + 0.9\*p2;  plot(x, p, type="l");  ``` |

### B1.5. The mixture model

+ Given the k weight values  and k probability distribution functions , we can create the mixture model by the following



+ In the matrix form, we can write



### B1.6. Generalized weights

+ You need to make sure that the sum of  to be one. If not, you need to normalize the vector by dividing its entries with the sum. In fact, you can write the mixture model by the following general formula with any non-negative vector 



# B2. The k-means procedure

### B2.1. The compression motivation

|  |  |  |
| --- | --- | --- |
|  |  |  |
| The original dataset | The compressed dataset from 5 dimensions to 1 dimension | After reconstruction, we will have new vectors differences from the previous vector, but they are close enough |

### B2.2. We can measure the errors (differences) between the vectors

|  |  |
| --- | --- |
|  | Figure: the reconstruction error subject to the vocabulary |

### B2.3. Comparing two vocabularies

+ We are given the dataset D and two vocabularies C1 and C2.

+ We want to know which vocab is better for the dataset D.

+ We have to use error to test

|  |  |
| --- | --- |
|  | C1 =    C2 = |

+ We encode and decode and measure the reconstruction error

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

+ The total error is 71 for C1 and 41 for C2. We can conclude that C2 is better for D.

### B2.4. The k-mean procedure

+ The whole procedure starts with

|  |
| --- |
| [L, C] <- kmeans(X, kpar)  {  C = random(kpar, ncol(X));  L = label(X, C);  for(s=0; s<=maxstep; s++)  {  C = average(L, X, kpar);  L = label(X, C);  }  return(L, C);  } |

+ We can create randomized vector by any random functions in programming languages.

+ The labeling step: assuming that we have the list of centers or means  we need to give each xj one label zj by selecting the nearest center.

We have to compute  and we choose 

+ The averaging step (M-step): assuming that we have the labels, we can use the label to compute the centers



### B2.5. RMD code

|  |
| --- |
| # To show how k-means procedure works  ```{r}  rm(list=ls());  library(nnet); library(ggplot2);  ```  ## 1. Labeling step  ```{r}  kmeanLabel <- function(X, C)  {  rows <- nrow(X);  cols <- nrow(C);  labels <- 1:rows;  zj <- 1:cols;  for(j in 1:rows)  {  for(k in 1:cols) zj[k] <- sum(abs( X[j, ] - C[k, ] ));  labels[j] <- which.min(zj);  }  return(labels);  }  ```  ## 2. Averaging step  ```{r}  kmeanAvg <- function(X, L, kpar)  {  rows <- nrow(X);  C <- matrix(nrow=kpar, ncol=ncol(X), data=0);  N <- rep(0, ncol(X));  for(j in 1:rows)  {  zj <- L[j];  C[zj, ] <- C[zj, ] + as.numeric( X[j, ] );  N[zj] <- N[zj] + 1;  }  for(k in 1:kpar) if(N[k] > 0) C[k, ] <- C[k, ] / N[k];  return(C);  }  ```  ## 3. Initializing step  ```{r}  kmeanInit <- function(X, kpar)  {  d <- ncol(X);  C <- matrix(nrow=kpar, ncol=d);  for(k in 1:kpar) C[k, ] <- runif(d, 0, 100);  return(C);  }  ```  ## 4. The whole procedure  ```{r}  kmeanProc <- function(X, kpar, ms)  {  C <- kmeanInit(X, kpar);  L <- kmeanLabel(X, C);  for(s in 1:ms)  {  C <- kmeanAvg(X, L, kpar);  L <- kmeanLabel(X, C);  }  return( list(L, C) );  }  ```  ## 5. An example  ```{r}  X <- data.frame(rbind(  c(121, 0, 0, 0, 0), c(122, 0, 0, 0, 0),  c(123, 0, 0, 0, 0), c(124, 0, 0, 0, 0),  c(125, 0, 0, 0, 0), c(0, 0, 231, 0, 0),  c(0, 0, 232, 0, 0), c(0, 0, 233, 0, 0),  c(0, 0, 234, 0, 0), c(0, 0, 235, 0, 0),  c(0, 0, 0, 0, 432), c(0, 0, 0, 0, 433),  c(0, 0, 0, 0, 434), c(0, 0, 0, 0, 435),  c(0, 0, 0, 0, 436), c(0, 0, 0, 0, 437)  ));  M <- kmeanProc(X, 4, 20);  ``` |

# B3. The E-M procedure (expectation maximization)

### B3.1. Extension to many labels

+ For each sample xj, we will have k labels for corresponding k centers (means) with the following

+ The label can be a real number to denote how much xj belongs to ck. For example, if we have xj with zj = (c1-> 0.5, c2->0.5, c3->0), this means xj half-belongs to c1 and half-belongs to c2 and does not belong to c3.

### B3.2. The E-step / labeling step

+ In the labeling step, we will give k labels for each sample xj.



+ in which each sim(xj, ck) is the similarity between xj and ck. We can choose  for example. The max value will be 1 when x=c. When x is far away from c, sim(x, c) will approach zero.

+ We have to normalize the label in order to have the measure within [0, 1] interval for comparison and combination later.



### B3.3. The M-step / averaging step

+ After we have all the labels, we know well which sample belongs to which centers. Now we can update the centers by averaging the



### B3.4. The whole procedure

+ Randomize the centers (c1, c2, .., ck)

+ For each xj, we create the label vector zj = (zj1, zj2, .., zjk) from xj to all centers (c1, c2, .., ck) and we normalize zj

repeat

{

+ average the centers (c1, c2, .., ck) when we have labels Z from previous step

+ For each xj, we create the label vector zj = (zj1, zj2, .., zjk) from xj to all centers (c1, c2, .., ck) and we normalize zj

}

# B4. Gaussian mixture model

### B4.1. Motivation

+ A Gaussian mixture model is a mixture model (linear combination of Gaussian components).

+ GMM is suitable to describe a class with many Gaussian (elliptic) subclasses. Notice that when we use k-means to divide the space into the subclasses, we will have different topology (Voronoi diagram) instead of the elliptic balls.

|  |  |
| --- | --- |
|  | Figure: when we want to model the face concept (class) with two elliptic subclasses, we have to use GMM  When we want elliptic sub classes of the data, we can use GMM |

### B4.2. The Gaussian mixture model

+ We can compute



+ In which, each gaussian component can be computed by the following formula



+ For example in two dimension we can have



### B4.3. Learning GMM with EM method

+ We have to learn the parameters  with EM method.

+ In the E-step, we try to label each point xj with all centers by the gaussian components.



+ Then we have to normalize zj as well

+ In the M-step, we have to estimate (w1, m1, s1, .., wk, mk, sk) from the label we have

