統計學習初論（106-2）

作業四

作業設計：盧信銘
國立台灣大學資管系

截止時間：2018 年 4 月 24 日上午 9 點

第一題請至 RSAND 上批改，第 a 小題範例命令為
sl_check_hw4q1a ./your_program，第 b 小題為：
sl_check_hw4q1b ./your_program。

第二題請至 RSAND 上批改，第 a 小題範例命令為
sl_check_hw4q2a ./your_program，第 b 小題為：
sl_check_hw4q2b ./your_program。

作業自己做。嚴禁抄襲。不接受紙本繳交，不接受遲交。請以英文或中文作答。

第一題

(50 points) We are going to explore the problem of identifying smartphone position through probabilistic generative models. Motion sensors in smartphones provide valuable information for researchers to understand its owners. An interesting (and more challenging) task is to identify human activities through the data recorded by motion sensors. For example, we want to know whether the smartphone owner is walking, running, or biking. In this homework problem, we are going to tackle a simpler problem. We want to know the static position of the smartphone. There are six possible positions:

- Phoneonback: The phone is laying on the back of the phone with the screen pointing up (away from the ground).
- Phoneonfront: The phone is laying on the back of the phone with the screen pointing towards the ground
- Phoneonbottom: The phone is standing on the bottom of the screen, meaning the bottom is pointed towards the ground
- Phoneontop: The phone is standing on the top of the screen, meaning the top is pointed towards the ground
- Phoneonleft: The phone is laying on the left side of the screen.
- Phoneonright: The phone is laying on the right side of the screen.

The input data is the reading of the accelerometer (cf. https://en.wikipedia.org/wiki/Accelerometer) in the smartphone. We have a training

dataset that contains about 28,500 data points for phones in each of the six positions. The following is the first six data points for the first position (Phoneonback):

```
> head(traindata[[1]])
          x            y        z
1 0.1388092  0.074340820 9.801056
2 0.1649933  0.006500244 9.690369
3 0.2114105 -0.001831055 9.755829
4 0.1840363 -0.007781982 9.774872
5 0.1423798  0.005310059 9.765350
6 0.1852264  0.026733398 9.829620
```

We are going to train a model that can predict smartphone positions given its accelerometer readings. To achieve this goal, we are going to divide this question into two sub-questions. The first is about training a generative classifier, and the second is to predict smartphone positions.

(a) (25 points) To train probabilistic generative classifiers, we need to assume how the data are generated in one given position. Here we are going to adopt a simple assumption that the data are generated from multivariate Gaussian distributions. To train a model under this assumption, we need to compute the mean and variance of the readings from the training data in phonetrain.rdata. This data file contains two variables. The first variable is outclass, which is a vector of strings of possible phone positions:

```
> outclass
[1] "Phoneonback"   "Phoneonbottom" "Phoneonfront"  "Phoneonleft"
"Phoneonright"
[6] "Phoneontop"
```

The second variable is traindata, which is a list of length 6. Each component is a data frame that are training data for a particular position. The 6 components in traindata have the same order as the strings in outclass. That is, traindata[[1]] is for Phoneonback, traindata[[2]] is for Phoneonbottom, and so on. For the data frame in each position, compute the mean (named mu1), covariance (use unbiased estimator; named sigma1), precision (the inverse of covariance; named prec1), logarithm of determinant of sigma1 (named detsig_log), and number of observations in this data frame (named N1). An example command for the construction of this list data structure is:

```
list(mu1=mu1, sigma1=cov1, prec1=solve(cov1), detsig_log=logdet, N1=N1)
```

The trained model is a list of six components, ordered by the position strings in outclass. Each component is a list of parameters for a position as explained above.

Write a function named pgm_train for this task. This function takes two input parameters. The first is outclass, and the second is alldata, which is the list of six data frames for the training data of different positions. This function should return the trained model as explained above.

Input and output, sample 1

```
> setwd('your_path')
> load('phonetrain.rdata')
>
> train2 = list()
> for(aclass in outclass) {
```

```
+     train2[[aclass]] = traindata[[aclass]][1:500,]
+ }
> model1=pgm_train(outclass, train2)

> model1[1]
$Phoneonback
$Phoneonback$mu1
          x           y           z
0.16199402 0.01477441 9.73784279

$Phoneonback$sigma1
              x             y             z
x  6.382636e-04 -6.211061e-05  1.310078e-05
y -6.211061e-05  6.812823e-04 -3.367878e-05
z  1.310078e-05 -3.367878e-05  1.001629e-03

$Phoneonback$prec1
           x          y          z
x 1581.02638  143.35381  -15.85886
y  143.35381 1483.26227   47.99823
z  -15.85886   47.99823 1000.19502

$Phoneonback$detsig_log
  modulus
-21.56515

$Phoneonback$N1
[1] 500


> model1[3]
$Phoneonfront
$Phoneonfront$mu1
          x           y           z
 0.08098285  0.31618890 -9.75056005

$Phoneonfront$sigma1
              x             y             z
x  6.044951e-04 5.133011e-06 -6.190712e-05
y  5.133011e-06 6.353561e-04  1.661259e-06
z -6.190712e-05 1.661259e-06  1.022019e-03

$Phoneonfront$prec1
          x          y         z
x 1664.7188  -13.71290 100.85993
y  -13.7129 1574.04022  -3.38919
z  100.8599   -3.38919 984.57062

$Phoneonfront$detsig_log
  modulus
-21.66472

$Phoneonfront$N1
[1] 500


> model1[5]
$Phoneonright
$Phoneonright$mu1
          x           y           z
-9.65647737  0.15743481 -0.07427301
```

```
$Phoneonright$sigma1
                x             y             z
x  6.221814e-04 -1.944180e-05  1.720659e-05
y -1.944180e-05  6.040970e-04 -2.081885e-05
z  1.720659e-05 -2.081885e-05  1.090656e-03

$Phoneonright$prec1
           x          y         z
x 1609.51604   50.95788 -24.41962
y   50.95788 1658.06633  30.84586
z  -24.41962   30.84586 917.85355

$Phoneonright$detsig_log
 modulus
-21.6171

$Phoneonright$N1
[1] 500
```

Input and output, sample 2

```
> train3 = list()
> for(aclass in outclass) {
+     train3[[aclass]] = traindata[[aclass]][1:1500,]
+ }
> model2=pgm_train(outclass, train3)
> model2[2]
$Phoneonbottom
$Phoneonbottom$mu1
        x         y         z
0.1444420 9.8831614 0.1100169

$Phoneonbottom$sigma1
                x             y             z
x  6.624064e-04 -1.087213e-07 -4.537591e-06
y -1.087213e-07  6.420947e-04 -2.331090e-06
z -4.537591e-06 -2.331090e-06  1.004589e-03

$Phoneonbottom$prec1
              x            y          z
x 1509.6939263    0.2803843   6.819734
y    0.2803843 1557.4159359   3.615160
z    6.8197338    3.6151603 995.471607

$Phoneonbottom$detsig_log
  modulus
-21.57362

$Phoneonbottom$N1
[1] 1500


> model2[4]
$Phoneonleft
$Phoneonleft$mu1
          x           y           z
 9.89267651  0.27117529 -0.02911975

$Phoneonleft$sigma1
```

```
                x              y              z
x  6.720721e-04 -1.761831e-05 -9.695159e-07
y -1.761831e-05  6.633099e-04  3.235197e-05
z -9.695159e-07  3.235197e-05  1.040748e-03

$Phoneonleft$prec1
             x          y           z
x 1488.9724017   39.54120   0.1579119
y   39.5412017 1510.93011 -46.9308837
z    0.1579119  -46.93088 962.3061929

$Phoneonleft$detsig_log
  modulus
-21.49344

$Phoneonleft$N1
[1] 1500


> model2[6]
$Phoneontop
$Phoneontop$mu1
          x          y          z
 0.03439581 -9.54268689 -0.27299432

$Phoneontop$sigma1
                x              y              z
x  6.796703e-04  1.472893e-05 -4.098093e-06
y  1.472893e-05  6.685889e-04 -2.093863e-05
z -4.098093e-06 -2.093863e-05  1.013832e-03

$Phoneontop$prec1
            x          y         z
x 1472.03272  -32.26322   5.28389
y  -32.26322 1497.36257  30.79454
z    5.28389   30.79454 987.01366

$Phoneontop$detsig_log
  modulus
-21.49941

$Phoneontop$N1
[1] 1500
```

Evaluation: All credits will be given based on the correctness of 10 testing cases. Correct output in a case is worth 2.5 points.

(b) (25 points) To predict phone positions, compute the posterior of each position given the trained model, and select the position with the largest probability. That is, given the reading $g = (g_x, g_y, g_z)^T$, the probability that the phone position is k can be written as $Prob(POS = k|g) \propto Prob(g|POS = k)Prob(POS = k)$.

We do not prefer any position so a reasonable setting is $Prob(POS = k) = c$, for $k = 1, 2, \dots, 6$. Here c is a constant, and this constant plays no role in influencing the prediction outcome.

As for $Prob(g|POS = k)$, since we adopted Gaussian assumption in this problem, $Prob(g|POS = k) = N(g|\mu_k, \Sigma_k)$, where μ_k and Σ_k are learned from the training data. Let $q_k = Prob(g|POS = k)$, then the prediction of phone position can be determined by looking at $q = (q_1, q_2, q_3, q_4, q_5, q_6)$, and select the one with the largest value.

Write a function named pgm_predict, This function takes two parameters. The first parameter, amodel, is the data structure of the trained model. The second parameter, testdata, is a data frame or matrix that contains accelerometer readers to be processed. You should check testdata and make sure it contains three columns. Return NULL if this condition is not satisfied.

Predict the phone position for data points in each row. Return a vector that contain the predicted phone position corresponding to testdata. You should use 1, 2, 3, 4, 5, and 6 to represent predicted cell phone positions. These integers corresponds to the strings in outclass.

Sample input and output:

```
> load('phonetrain.rdata')
> load('phonetest1.rdata')
>
> model1=pgm_train(outclass, traindata)
> pred1=pgm_predict(model1, testds1_feature)
>
>
>
> pred1[1:50]
 [1] 3 1 1 3 2 3 6 2 6 4 3 1 5 2 1 3 4 2 2 4 4 3 1 3 5 5 2 4 6 5 4
3 2 5 2 5 3 3 5 4 4 3 1
[44] 3 1 3 5 3 5 5
```

Evaluation: All credits will be given based on the correctness of 10 testing cases. Correct output in a case is worth 2.5 points.

第二題

(50 points) The goal of sentence categorization is to assign a sentence to one of two or more predefined categories. We are going to develop a sentence classification model using regularized logistic regression. To tackle this homework, you need to understand the concept of Laplace approximation and evidence maximization for logistic regression. You should watch the online video for this topic: https://www.youtube.com/playlist?list=PLvVPTibZrkBpC2Ga8xizhOH1ObiYz8wzq

The slides can be download from Ceiba (@Week7).

The underlying problem is to detect sentences that mentioned about cost reduction efforts during the current fiscal year. We denoted this category as “O.COST.” A sentence can belong to O.COST, or not belongs to this category.

As an illustrative example, the following sentences belong to O.COST:

- The Company expects the operational restructuring plan to be fully implemented by June 30, 2006, and the implementation of the operational restructuring plan will result in severance related and other charges of approximately $14.6 million.
- Operational Restructuring On August 19, 2005, the Company announced that it had taken the initial step in implementing an expense restructuring plan, necessitated by the Company's declining revenue trend over the previous two and one-half years.

The following sentences are examples not belonging to O.COST:

- Such capital expenditures and employee compensation costs will continue to be incurred in advance of the first revenues to be earned from the contract, expected later in 2006.
- The Company expects many clients to continue to use their own internal recovery audit functions as a substitute for its recovery audit services.

The o_cost_train.rdata file contains a data frame named ds4a_train. Each row represents a sentence. To save your time, all sentences have been process via natural language processing approaches and have been converted to appropriate representations. The first column (named o.cost) is the label of the sentence. The value is either "pos" or "neg." The "pos" string indicates that this sentence belong to the O.COST category. There are three types of feature representations in this dataset. We summarize the information below:

Column Name	Column Position	Description	Value
f_past	2	Sentence timing	0 or 1
g1 to g100	3-102	Latent topics	0 to 1
w*	103-3102	Unigram (word feature)	0 or 1

The specific feature representations used in this dataset is beyond the scope of this course. You only need to know the column positions of the three types of features in order to complete the following two tasks.

(a) (25%) Write a function named logicreg_l2_train to perform model training. This function should take the following input parameters (in this order):
   1. y: the (one-column) matrix of outcome value. The value should be either 1 or 0, representing positive and negative cases.
   2. xmat: the matrix containing feature values. Should be compatible with y.
   3. lambda_rate: the coefficient to compute initial lambda value. The default value is 0.0005.
   4. param_tol: the tolerance of error. The default value is 10^{-5}.
   5. granditertol: the tolerance of error for overall convergence, measured by number of inner iteration. The default value is 2.
   6. outitermax: the maximum number of outer iteration. The default value is 50.
   7. inneritermax: the maximum number of inner iteration. The default value is 20.

8. debuglevel: whether to output debug information. The default value is 0. You should not output any debug information when the value is 0. Set this value to a positive integer if you want to output debugging information when developing the function.

The function should start by computing the initial value for subsequent numerical optimization procedure. Set initial $\lambda = lambda\_rate \times N$, where N is the number of training instance, and we can compute the initial w by:

$$w = (\lambda I + xmat^T xmat)^{-1} xmat^T y.$$

We are ready to iterate to update w and λ. The outer iteration can loop for a maximum of *outitermax* times. In each outer iteration, we update w using Newton's Method, and then update λ through the algorithm presented in class. We called the updating of w and λ the "inner iteration."

To update w, compute the gradient and hessian matrix according to our discussion in class (cf. slides 25, 05c linear model for classification.pdf), and then compute the new w via $w^{(new)} = w^{(old)} - H^{-1}\nabla E(w)$. Declare convergence if the mean absolute difference between the new and old w is less than param_tol. This inner iteration should loop for a maximum of inneritermax times.

To update λ, follow our discussion in class to compute the new λ given the new w (cf. slide 30, 05c_online linear model for classification.pdf). Declare convergence if the mean absolute difference between the new and old λ is less than param_tol. This inner iteration should loop for a maximum of inneritermax times.

After updating w and λ, continue the outer iteration if the inner iteration for w takes more than 2 (granditertol) iterations to converge. Otherwise, break the outer iteration and return a list that contain the parameters of the trained model.

The returned list should include the following components (in this order):

- w: the estimated coefficients
- w_sd: the standard deviation of the estimated coefficients. Recall that the posterior of w is approximately normal with a covariance S_N, and $S_N^{-1} = S_0^{-1} + \sum_{i=1}^{n} y_n(1-y_n)\phi_n\phi_n^T = \lambda I + \Phi^T R \Phi$. This component (w_sd) is simply the square root of the diagonal elements of S_N.
- lambda: The lambda coefficient.
- M: the dimension of input features (i.e., the length of w).
- N: number of training instances.

Sample input and output

```
> load(file="o_cost_train.rdata")

#Sample 1
> dm_train_t = as.numeric(ds4a_train[,1] == "pos")
> tall=as.matrix(dm_train_t)
> xmat = model.matrix(~f_past+g1+g2+g3+g4+g5+g6+g7+g8+g9+g10,
data=ds4a_train[,-1])
> model1 = logicreg_l2_train(tall, xmat, debuglevel=0)
> model1
$w
                  [,1]
(Intercept) -1.4053791
f_past      -2.1991748
g1           0.8093526
```

```
g2          -8.5513319
g3          -6.9447606
g4           4.5614423
g5           0.3788263
g6           2.4862296
g7          -4.8258687
g8          -4.5343458
g9          -9.7827331
g10         -1.2796956

$w_sd
(Intercept)      f_past          g1          g2          g3          g4
 0.04366356  0.23233368  0.65868466  2.45398092  2.02103128  0.49332275
         g5          g6          g7          g8          g9         g10
 0.44533757  0.72333805  1.60224100  1.41466936  2.48016361  1.25627118

$lambda
[1] 0.03738331

$M
[1] 12

$N
[1] 6106


#Sample 2
> dm_train_t = as.numeric(ds4a_train[,1] == "pos")
> tall=as.matrix(dm_train_t)
> xmat2 = model.matrix(~., data=ds4a_train[,c(2, 103:204)])
> model2 = logicreg_l2_train(tall, xmat2, debuglevel=0)
> print(head(model2$w, n=10))
                       [,1]
(Intercept)       -3.609966
f_past            -3.251449
w4035_reduction   11.494339
w4037_restructure  9.834049
w4078_cost         6.479697
w3998_employee     8.264094
w4111_costs        2.953915
w3492_savings     11.699770
w3755_severance   10.435952
w4028_reduce      10.774457
> print(head(model2$w_sd, n=10))
      (Intercept)            f_past   w4035_reduction w4037_restructure
        0.1133109         0.4242634         1.0368524         0.8800527
       w4078_cost    w3998_employee       w4111_costs     w3492_savings
        1.1815602         1.0729767         1.0726329         1.2645677
  w3755_severance       w4028_reduce
        1.1358556         1.0819511
> model2$lambda
[1] 0.03569337
> model2$M
[1] 104
> model2$N
[1] 6106
```

Evaluation: All credits will be given based on the correctness of 10 testing cases. Correct output in a case is worth 2.5 points.

(b) (25%) Write a function named logicreg_l2_predict to perform prediction on a trained model. Use the learned parameter *w* to perform prediction. This function takes the following parameters (in this order):

1. model1: The learned model.
2. xmat: the feature matrix to be used for prediction.

This function should return a list that contains the following component (in this order):

- prob: a vector of predicted probability for each sentence.
- class: a vector of predicted class (0 or 1) for each sentence.

Sample input and output:

```
#Sample 1
> load(file="o_cost_train.rdata")
> load(file="o_cost_test.rdata")
>
> dm_train_t = as.numeric(ds4a_train[,1] == "pos")
> tall=as.matrix(dm_train_t)
> xmat = model.matrix(~f_past+g1+g2+g3+g4+g5+g6+g7+g8+g9+g10,
data=ds4a_train[,-1])
> model1 = logicreg_l2_train(tall, xmat, debuglevel=0)
> #perform prediction
> xmattest1 = model.matrix(~f_past+g1+g2+g3+g4+g5+g6+g7+g8+g9+g10,
data=ds4a_train[,-1])
> logicpred1 = logicreg_l2_predict(model1, xmattest1)
> head(logicpred1$class,n=25)
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
> head(logicpred1$prob,n=25)
          [,1]
126  0.2448322
234  0.2116120
235  0.2054513
236  0.1935176
256  0.2094563
522  0.1918210
643  0.1883066
645  0.1889833
662  0.1975938
712  0.1893408
739  0.2117428
776  0.1858589
894  0.1851180
897  0.1813298
898  0.1780135
899  0.1673133
2360 0.1702156
2361 0.2426719
2397 0.1872902
2442 0.2024119
2457 0.1956855
2458 0.1745073
2510 0.2607178
2513 0.1967121
2514 0.1066816


#Sample 2
```

```
> dm_train_t = as.numeric(ds4a_train[,1] == "pos")
> tall=as.matrix(dm_train_t)
> xmat2 = model.matrix(~., data=ds4a_train[,c(2, 103:204)])
> model2 = logicreg_l2_train(tall, xmat2, debuglevel=0)
> #perform prediction
> xmattest2 = model.matrix(~., data=ds4a_test[,c(2, 103:204)])
> logicpred2 = logicreg_l2_predict(model2, xmattest2)
> head(logicpred2$class,n=30)
 [1] 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0
> head(logicpred2$prob,n=30)
           [,1]
266  0.99441301
644  0.99508324
815  0.04801758
2509 0.98630634
2511 0.96278682
2523 0.99776405
3168 0.99999183
3715 0.99990698
3716 0.36628387
3765 0.60906632
4286 0.97563708
4709 0.63226300
4895 0.91296762
4937 0.97449792
5989 0.99996757
6091 0.99982256
6154 0.96100325
6165 0.97107921
6174 0.96185452
6293 0.07284124
6384 0.99954049
6410 0.99994020
6529 0.95066773
6546 0.99994284
6551 0.99418181
6560 0.98777522
6661 0.99868930
6662 0.99916273
6665 0.99481762
7911 0.34222445
```

Evaluation: All credits will be given based on the correctness of 10 testing cases. Correct output in a case is worth 2.5 points.