ORF525, Assignment 3, Problem 6

Igor Silin

(a)

Let's just execute the code from *human.r*:

```
library("png")

crop.r=function(X, h, w){
    h1=dim(X)[1]
    w1=dim(X)[2]
    x1=sample(1:(w1-w+1),1)
    y1=sample(1:(h1-h+1),1)
    return(X[(y1:(y1+h-1)), (x1:(x1+w-1))])
}

crop.c=function(X, h, w){
    h1=dim(X)[1]
    w1=dim(X)[2]
    h.margin=floor((h1-h)/2)
    w.margin=floor((w1-w)/2)
    x1=w.margin+1
    y1=h.margin+1
    return(X[(y1:(y1+h-1)), (x1:(x1+w-1))])
}

hog=function(xgrad, ygrad, hn, wn, an){
    h=dim(xgrad)[1]
    w=dim(xgrad)[2]
    h1=h/hn
    w1=w/wn
    xr=(1:wn)*w1
    xl=xr-(w1-1)
    yd=(1:hn)*h1
    yu=yd-(h1-1)
    theta=ifelse(ygrad>0, acos(xgrad/sqrt(xgrad^2+ygrad^2)),
                 -acos(xgrad/sqrt(xgrad^2+ygrad^2)))
    angle=c()
    for (i in 1:hn){
        for (j in 1:wn){
            angle=c(angle, hist(as.vector(theta[yu[j]:yd[j], xl[i]:xr[i]])+pi,
                                breaks=seq(from=0, to=2*pi, by=2*pi/an), plot=F)$counts/(h1*w1))
        }
    }
    return(angle)
}

grad=function(X, h, w, pic){
    X1=crop.c(X, h+2, w+2)
    xgrad=(X1[-c(1,h+2), -c(1, 2)]-X1[-c(1,h+2), -c(w+1,w+2)])/2
    ygrad=(X1[-c(h+1,h+2), -c(1,w+2)]-X1[-c(1, 2), -c(1, w+2)])/2
```

```
    if (pic==TRUE){
        plot(c(),c(), asp=1, xlim=c(0,70), ylim=c(0,130), xlab="X", ylab="Y")
        for (i in 1:h){
            for (j in 1:w){
                arrows(x0=j, y0=h+1-i, x1=j+xgrad[i,j]*5, y1=h-i+1+ygrad[i,j]*5, length=0.01)
            }
        }
    }
    return(list("xgrad"=xgrad, "ygrad"=ygrad))
}

library(png)
npos=500 # number of pictures with humans
nneg=500 # number of pictures without humans
h=128 # height of the central part
w=64 # width of the central part
hn=4 # number of partitions along the height
wn=4 # number of partitions along the width
an=6 # number of partitions on [0, 2pi]
Xpos=matrix(rep(0,an*hn*wn*npos), nrow=npos, ncol=an*hn*wn)
Xneg=matrix(rep(0,an*hn*wn*nneg), nrow=nneg, ncol=an*hn*wn)

set.seed(525)


current_folder = getwd()

# loading data
print('load images with positive labels')
```

```
## [1] "load images with positive labels"
```

```
setwd(paste0(current_folder, "/pictures/pos"))
posnames=list.files(pattern="*.png") # get all the names of files
                                     # in the specified location as a list of strings

for (i in 1:npos){
    X=readPNG(posnames[i])
    gf=grad(X, h, w, F) # generate gradient field
    Xpos[i,]=hog(gf$xgrad, gf$ygrad, hn, wn, an) # extract features
    if (i %% 50 == 0)
    {
      cat('\nloading ', i/npos*100, '% of images.')
    }
}
```

```
##
## loading  10 % of images.
## loading  20 % of images.
## loading  30 % of images.
## loading  40 % of images.
## loading  50 % of images.
## loading  60 % of images.
## loading  70 % of images.
```

```
## loading  80 % of images.
## loading  90 % of images.
## loading  100 % of images.
```

```
# load negative images
cat("\n\nload images with negative labels\n")
```

```
##
##
## load images with negative labels
```

```
setwd(paste0(current_folder, "/pictures/neg"))
negnames=list.files(pattern="*.png")
for (i in 1:nneg){
    X=readPNG(negnames[i])
    gf=grad(X, h, w, F)
    Xneg[i,]=hog(gf$xgrad, gf$ygrad, hn, wn, an)
    if (i %% 50 == 0)
    {
      cat('\nloading ', i/npos*100, '% of images.')
    }
}
```

```
##
## loading  10 % of images.
## loading  20 % of images.
## loading  30 % of images.
## loading  40 % of images.
## loading  50 % of images.
## loading  60 % of images.
## loading  70 % of images.
## loading  80 % of images.
## loading  90 % of images.
## loading  100 % of images.
```

```
X.all=rbind(Xpos, Xneg) # Combine all the data together
Y.all=c(rep(1, npos), rep(0, nneg))   #Create binary response variable

#move back to  working folder
setwd(current_folder)
```

(b)

i.

Let's shuffle and divide the data into training and testing parts:

```
set.seed(525)

N.all = dim(X.all)[1]
N.train = 4/5*N.all
N.test = 1/5*N.all

permutation = sample(N.all, replace=FALSE)
```

```
X.train = X.all[permutation[1:N.train], ]
X.test = X.all[permutation[(N.train+1):N.all], ]
Y.train = Y.all[permutation[1:N.train]]
Y.test = Y.all[permutation[(N.train+1):N.all]]

Data.train = data.frame(Y=Y.train, X.train)
Data.test = data.frame(X.test)
```

ii.

Let's fit the training data:

```
fitted1 = glm(Y ~ ., family=binomial, data=Data.train)
```

iii.

Let's predict on the testing data and report the misclassification rate/testing error of $fitted1$:

```
Linear.pred = predict(fitted1, newdata=Data.test)
Prob.pred = exp(Linear.pred)/(1+exp(Linear.pred))
Y.pred = as.numeric(Prob.pred > 0.5)

error = sum(Y.pred != Y.test)/N.test
sprintf('iii. fitted1: Misclassification rate/Testing error = %f', error)
```

```
## [1] "iii. fitted1: Misclassification rate/Testing error = 0.150000"
```

iv.

Let's select features uning *step* function (output is huge, so let's omit it):

```
fitted2 = step(fitted1)
```

Let's compare two models:

```
summary(fitted1)
```

```
##
## Call:
## glm(formula = Y ~ ., family = binomial, data = Data.train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1645  -0.0205   0.0000   0.0496   2.8986
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -31.1407    16.3460  -1.905   0.0568 .
## X1           11.3112    10.1941   1.110   0.2672
## X2            7.5824     8.2727   0.917   0.3594
## X3           23.3603     9.7871   2.387   0.0170 *
## X4           -0.6755     7.6465  -0.088   0.9296
## X5           19.3781     9.5873   2.021   0.0433 *
## X6           -1.7121     8.7412  -0.196   0.8447
```

```
## X7          12.5949    13.7067   0.919   0.3582
## X8          17.0124    12.8431   1.325   0.1853
## X9          10.9110    14.1182   0.773   0.4396
## X10         13.0829    11.2833   1.159   0.2463
## X11         13.6592    13.4432   1.016   0.3096
## X12         25.8770    11.9157   2.172   0.0299 *
## X13        -18.4119    12.4294  -1.481   0.1385
## X14          2.1123    11.7503   0.180   0.8573
## X15        -20.5486    12.2142  -1.682   0.0925 .
## X16          6.3066    11.4174   0.552   0.5807
## X17        -18.2523    11.6241  -1.570   0.1164
## X18        -11.8980    11.1116  -1.071   0.2843
## X19         11.3650     7.8650   1.445   0.1485
## X20          1.0001     6.8927   0.145   0.8846
## X21          4.7487     8.0766   0.588   0.5566
## X22         -1.4240     8.6396  -0.165   0.8691
## X23         -1.4699     8.9315  -0.165   0.8693
## X24        -15.6035     8.8441  -1.764   0.0777 .
## X25          8.9993    10.1436   0.887   0.3750
## X26          7.3395     9.5828   0.766   0.4437
## X27          9.2999     9.3357   0.996   0.3192
## X28         -7.4462     9.7259  -0.766   0.4439
## X29          6.0406     9.5437   0.633   0.5268
## X30          7.7079     8.9884   0.858   0.3911
## X31        -20.7408     9.1896  -2.257   0.0240 *
## X32        -13.2696    10.3777  -1.279   0.2010
## X33         -8.8758     9.1190  -0.973   0.3304
## X34          5.9338     8.2980   0.715   0.4746
## X35        -16.5491     9.5383  -1.735   0.0827 .
## X36          7.2597     7.8175   0.929   0.3531
## X37         28.5553    18.2807   1.562   0.1183
## X38          2.4593    18.1766   0.135   0.8924
## X39         12.9141    17.3864   0.743   0.4576
## X40          2.6394    17.3529   0.152   0.8791
## X41        -10.6240    17.6227  -0.603   0.5466
## X42         22.1415    17.0892   1.296   0.1951
## X43        -19.1579    12.1451  -1.577   0.1147
## X44         -3.1318    12.2420  -0.256   0.7981
## X45        -31.5789    13.4556  -2.347   0.0189 *
## X46         -6.9843    12.0447  -0.580   0.5620
## X47        -31.3754    13.4718  -2.329   0.0199 *
## X48        -25.8021    12.6647  -2.037   0.0416 *
## X49          7.5409     9.5496   0.790   0.4297
## X50         -7.6797    10.9335  -0.702   0.4824
## X51         -0.5227     9.6252  -0.054   0.9567
## X52         10.4318    10.1855   1.024   0.3057
## X53          6.7056    10.0527   0.667   0.5047
## X54        -12.5473    10.7128  -1.171   0.2415
## X55        -14.8547    13.9780  -1.063   0.2879
## X56         -9.9427    14.2501  -0.698   0.4853
## X57        -15.3876    14.8267  -1.038   0.2993
## X58         15.8392    13.7511   1.152   0.2494
## X59         -8.3195    15.2812  -0.544   0.5861
## X60         19.2009    14.3879   1.335   0.1820
```

```
## X61           7.4610    14.4807   0.515   0.6064
## X62           0.8027    14.3743   0.056   0.9555
## X63          16.8579    13.6014   1.239   0.2152
## X64          25.6126    13.2342   1.935   0.0529 .
## X65           0.4109    14.2452   0.029   0.9770
## X66          18.1522    13.3469   1.360   0.1738
## X67          16.4571    14.5823   1.129   0.2591
## X68          16.3130    15.6252   1.044   0.2965
## X69          20.5675    15.9908   1.286   0.1984
## X70          11.2268    14.5859   0.770   0.4415
## X71          25.5007    16.0720   1.587   0.1126
## X72          19.0700    15.6988   1.215   0.2245
## X73          19.6039     9.8276   1.995   0.0461 *
## X74          13.9002     8.9917   1.546   0.1221
## X75          15.4295     9.6902   1.592   0.1113
## X76           5.1077     9.2696   0.551   0.5816
## X77          18.6547     9.1552   2.038   0.0416 *
## X78          15.8877     8.9237   1.780   0.0750 .
## X79         -34.5548    16.5352  -2.090   0.0366 *
## X80         -20.4230    14.9865  -1.363   0.1730
## X81         -30.9232    16.9198  -1.828   0.0676 .
## X82         -22.8770    15.0888  -1.516   0.1295
## X83         -28.8278    16.8757  -1.708   0.0876 .
## X84         -20.4683    15.2580  -1.341   0.1798
## X85          12.3846    13.8669   0.893   0.3718
## X86          36.5998    15.3903   2.378   0.0174 *
## X87          25.2827    14.6720   1.723   0.0849 .
## X88          11.5645    12.9789   0.891   0.3729
## X89          10.6535    13.1205   0.812   0.4168
## X90          11.0547    13.3582   0.828   0.4079
## X91          -5.4541    10.6925  -0.510   0.6100
## X92          -5.2594    10.5744  -0.497   0.6189
## X93          -2.1773     9.6930  -0.225   0.8223
## X94         -19.7347    10.1869  -1.937   0.0527 .
## X95         -17.2776    10.6992  -1.615   0.1063
## X96          -2.5675    10.6988  -0.240   0.8103
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1109.03  on 799  degrees of freedom
## Residual deviance:  194.25  on 703  degrees of freedom
## AIC: 388.25
##
## Number of Fisher Scoring iterations: 9
```

```
summary(fitted2)
```

```
##
## Call:
## glm(formula = Y ~ X1 + X2 + X3 + X5 + X8 + X10 + X12 + X13 +
##     X15 + X17 + X18 + X19 + X24 + X25 + X26 + X27 + X29 + X30 +
##     X31 + X32 + X33 + X35 + X37 + X41 + X42 + X43 + X45 + X47 +
##     X48 + X50 + X54 + X58 + X60 + X61 + X63 + X64 + X66 + X71 +
```

```
##     X73 + X74 + X75 + X77 + X78 + X79 + X86 + X87 + X94 + X95,
##     family = binomial, data = Data.train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.02487  -0.02830   0.00000   0.08197   3.02691
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -24.992      5.488  -4.554 5.26e-06 ***
## X1            17.335      5.559   3.119 0.001818 **
## X2             9.606      3.744   2.566 0.010296 *
## X3            26.125      5.642   4.630 3.65e-06 ***
## X5            22.142      5.647   3.921 8.83e-05 ***
## X8            11.463      4.423   2.591 0.009559 **
## X10            8.242      3.952   2.085 0.037038 *
## X12           17.254      5.316   3.245 0.001173 **
## X13          -16.752      4.868  -3.441 0.000579 ***
## X15          -16.654      5.142  -3.239 0.001201 **
## X17          -17.568      5.236  -3.355 0.000792 ***
## X18          -11.382      4.109  -2.770 0.005609 **
## X19            7.523      3.713   2.026 0.042745 *
## X24          -15.995      5.009  -3.193 0.001406 **
## X25            9.201      5.693   1.616 0.106047
## X26            9.997      5.538   1.805 0.071062 .
## X27            8.733      5.263   1.660 0.097014 .
## X29           11.570      5.792   1.998 0.045759 *
## X30           13.516      4.996   2.706 0.006820 **
## X31          -23.170      6.730  -3.443 0.000575 ***
## X32          -21.781      5.296  -4.113 3.90e-05 ***
## X33          -15.926      6.295  -2.530 0.011408 *
## X35          -14.642      6.818  -2.148 0.031745 *
## X37           16.149      4.410   3.662 0.000251 ***
## X41          -16.885      5.213  -3.239 0.001200 **
## X42           17.080      4.372   3.907 9.34e-05 ***
## X43           -9.844      4.237  -2.323 0.020169 *
## X45          -12.025      4.321  -2.783 0.005384 **
## X47          -19.145      4.982  -3.843 0.000121 ***
## X48          -14.178      5.208  -2.723 0.006478 **
## X50          -10.979      4.349  -2.524 0.011594 *
## X54          -17.510      5.034  -3.478 0.000504 ***
## X58           21.458      4.826   4.446 8.73e-06 ***
## X60           25.885      5.465   4.736 2.18e-06 ***
## X61            8.437      4.074   2.071 0.038381 *
## X63           16.482      4.328   3.809 0.000140 ***
## X64           24.201      4.775   5.068 4.01e-07 ***
## X66           17.331      4.550   3.809 0.000139 ***
## X71            6.660      4.199   1.586 0.112707
## X73           15.474      5.040   3.070 0.002140 **
## X74           10.358      3.650   2.838 0.004542 **
## X75            8.912      4.390   2.030 0.042322 *
## X77           13.397      4.381   3.058 0.002230 **
## X78           11.792      4.360   2.705 0.006835 **
## X79           -7.024      3.897  -1.802 0.071528 .
```

```
## X86           21.072      4.362   4.831 1.36e-06 ***
## X87           11.874      4.454   2.666 0.007682 **
## X94          -13.783      4.487  -3.072 0.002127 **
## X95          -10.290      4.141  -2.485 0.012947 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1109.03  on 799  degrees of freedom
## Residual deviance:  217.58  on 751  degrees of freedom
## AIC: 315.58
##
## Number of Fisher Scoring iterations: 9
```

We see that $fitted2$ has approximately twice less features than $fitted1$.

Let's predict on the testing data and report the misclassification rate/testing error of $fitted2$:

```
Linear.pred = predict(fitted2, newdata=Data.test)
Prob.pred = exp(Linear.pred)/(1+exp(Linear.pred))
Y.pred = as.numeric(Prob.pred > 0.5)

error = sum(Y.pred != Y.test)/N.test
sprintf('iv. fitted2: Misclassification rate/Testing error = %f', error)
```

```
## [1] "iv. fitted2: Misclassification rate/Testing error = 0.140000"
```

The quality improved a bit after feature selection.

v.

Let's fit the training data for the model with Lasso penaly called $fitted3$:

```
library("glmnet")

fitted3 = cv.glmnet(X.train, Y.train, family="binomial", alpha = 1, nfolds = 10)
```

Let's predict on the testing data and report the misclassification rate/testing error of $fitted3$:

```
Linear.pred = predict(fitted3, newx=X.test, s="lambda.min")
Prob.pred = exp(Linear.pred)/(1+exp(Linear.pred))
Y.pred = as.numeric(Prob.pred > 0.5)

error = sum(Y.pred != Y.test)/N.test
sprintf('v. fitted3: Misclassification rate/Testing error = %f', error)
```

```
## [1] "v. fitted3: Misclassification rate/Testing error = 0.115000"
```

This model with Lasso penalty outperforms the previous models.