MATH 988 Homework 1

Alexander Joyce, Andrew Pangia, Trevor Squires,
Maddy St. Ville, Peter Westerbaan

September 19, 2019

Introduction

We write a program that performs classifcation via any model and attempt to use our classifcation program to classify images of cats and dogs. We download the Kaggle cats and dogs dataset from Microsoft website (https://www.microsoft.com/en-us/download/details.aspx?id=54765). Since the images could be of different sizes and colours and some images may be corrupted and unreadable, we apply preprocessing to ignore the corrupted files and standardize the size of the pictures.

We attempt to work in both MATLAB and R, however, due to package installation issues in the Palmetto cluster, we only succeed in running MATLAB code on the cluster. We include the code run on the cluster, as well as the confusion matrix from the test set and the percent of correct predictions. For completeness, we also include the R code which we ran on our local machines and the corresponding results.

KNN

For classification, the following KNN program was written in R.

```
KNN.func<-function(X0,Y,Xo,k){
d<-sqrt(apply(t(t(Xo)-X0)^2,1,sum))
id<-order(d)[1:k]
yhat<-mean(Y[id])
return(yhat)
}
```

In the above R function, the inputs are defined as follows.

- `X0`: covariates associated with training set
- `Y`: responses associated with training set
- `Xo`: covariates associated with the test set (i.e., corresponding to the responses that are to be predicted)

- `k`: number of nearest neighbors

The function `KNN.func` returns the predicted response value, `yhat`.

Next, we consider applying KNN to the dogs vs cats image classification problem. After prepping the data and extracting feature matrices, we split the dataset into train and test sets: 90% of data went to training set and the other 10% went to the test set. We apply KNN to the testing data and examine accuracy rate for K=1, K=5, K=15, displayed in Table 1. K=15 nearest neighbors produced the best accuracy rate, though it is not adequate.

	K=1	K=5	K=15
Acc. Rate	.528	.539	.584

Table 1: Prediction accuracy rates for the test set.

		Image Label		
		Cat	Dog	Total
Predicted Label	Cat	971	760	1731
	Dog	279	490	769
	Total	1250	1250	2500

Table 2: Confusion matrix for KNN

The accuracy rating in 1 and the confusion matrix in 2 came from running R using a local machine. In addition, we also ran a KNN method in MATLAB using the cluster. The code for this can be found at http://www.github.com/trevorsquires9/math9880.

Transfer Learning

In addition to the naive KNN method applied above, we also attempted a more advanced technique in transfer learning. MATLAB's deep neural network toolbox has a pretrained alexnet neural network available. It has already been trained on over millions of images and can identify thousands of different object classes. Our approach is to simply utilize this pretrained network by retraining the last few layers on our kaggle dataset.

More specifically, Alexnet's first 25 layers were used as a type of feature extraction. Our dataset was converted into 4096x1 feature vectors via Alexnet's first few layers and then we trained the remaining 3 layers of the neural network to fine tune the classifications. The results were better than expected; we achieved 0.97 validation accuracy. The code can be found on github at http://www.github.com/trevorsquires9/math9880 under the appropriate subfolder. Table 3 showcases the confusion matrix for the size 7480 validation set.

		Image Label		
		Cat	Dog	Total
Predicted Label	Cat	3613	126	3739
	Dog	128	3613	3741
	Total	3741	3739	7480

Table 3: Confusion matrix for Transfer Learning Approach

Appendix

R Code - KNN

Data prep: follows from Data Preprocessing section of
https://rstudio-pubs-static.s3.amazonaws.com/236125_e0423e328e4b437888423d3821626d92.html.

```
# To install EBImage package:

source("https://bioconductor.org/biocLite.R")
biocLite("EBImage")
library(EBImage)
library(class)
library(caret)
library(pbapply)


path <- "C:\\Users\\owner\\Documents\\Clemson University\\ML Math 9810_9880\\"
image_dir <- "C:\\Users\\owner\\Documents\\Clemson University\\ML Math 9810_9880\\train"

# example.cat=readImage(file.path(image_dir, "cat.0.jpg"))
# display(example.cat)
#
# example.dog=readImage(file.path(image_dir, "dog.0.jpg"))
# display(example.dog)

width <- 28
height <- 28


extract_feature <- function(dir_path, width, height, is_cat = TRUE, add_label = TRUE) {
  img_size <- width*height
  ## List images in path
  images_names <- list.files(dir_path)
  if (add_label) {
    ## Select only cats or dogs images
    images_names <- images_names[grepl(ifelse(is_cat, "cat", "dog"), images_names)]
    ## Set label, cat = 0, dog = 1
```

```
    label <- ifelse(is_cat, 0, 1)
  }
  print(paste("Start processing", length(images_names), "images"))
  ## This function will resize an image, turn it into greyscale
  feature_list <- pblapply(images_names, function(imgname) {
    ## Read image
    img <- readImage(file.path(dir_path, imgname))
    ## Resize image
    img_resized <- resize(img, w = width, h = height)
    ## Set to grayscale
    grayimg <- channel(img_resized, "gray")
    ## Get the image as a matrix
    img_matrix <- grayimg@.Data
    ## Coerce to a vector
    img_vector <- as.vector(t(img_matrix))
    return(img_vector)
  })
  ## bind the list of vector into matrix
  feature_matrix <- do.call(rbind, feature_list)
  feature_matrix <- as.data.frame(feature_matrix)
  ## Set names
  names(feature_matrix) <- paste0("pixel", c(1:img_size))
  if (add_label) {
    ## Add label
    feature_matrix <- cbind(label = label, feature_matrix)
  }
  return(feature_matrix)
}

cats_data <- extract_feature(dir_path = image_dir, width = width, height = height)
dogs_data <- extract_feature(dir_path = image_dir, width = width, height = height,
is_cat = FALSE)

dim(cats_data)
dim(dogs_data)

saveRDS(cats_data, "cat.rds")
saveRDS(dogs_data, "dog.rds")

X1=as.matrix(cats_data[,-1])
Y1=as.vector(cats_data[,1])
X2=as.matrix(dogs_data[,-1])
Y2=as.vector(dogs_data[,1])

X=rbind(X1,X2)
Y=c(Y1,Y2)
```

```
complete.dat=data.frame(cbind(Y,X))
ind=unlist(createDataPartition(complete.dat$Y, p = .9, times = 1))
train.dat <- complete.dat[ind,]
dim(train.dat)

test.dat=complete.dat[-ind,]
dim(test.dat)


train.Y=train.dat$Y
factor.train.Y=as.factor(train.Y)
train.X=train.dat[,-1]


test.Y=test.dat$Y
factor.test.Y=as.factor(test.Y)
test.X=test.dat[,-1]

# KNN.func<-function(X0,Y,Xo,k){
#   d<-sqrt(apply(t(t(Xo)-X0)^2,1,sum))
#   id<-order(d)[1:k]
#   yhat<-mean(Y[id])
#   return(yhat)
# }

##########################################
# k=15
train.pred15=knn(train.X,train.X,train.Y,k=15)
train.err15=mean(train.pred15!=train.Y)
confusionMatrix(train.pred15,factor.train.Y)

t1=Sys.time()
test.pred15=knn(train.X,test.X,train.Y,k=15)
t2=Sys.time()
test.err15 <- mean(test.pred15!=test.Y)
confusionMatrix(test.pred15,factor.test.Y)

##########################################
# k=5
train.pred5=knn(train.X,train.X,train.Y,k=5)
train.err5=mean(train.pred5!=train.Y)
confusionMatrix(train.pred5,factor.train.Y)

t3=Sys.time()
test.pred5=knn(train.X,test.X,train.Y,k=5)
t4=Sys.time()
test.err5 <- mean(test.pred5!=test.Y)
```

```
confusionMatrix(test.pred5,factor.test.Y)

##########################################
# k=1
train.pred1=knn(train.X,train.X,train.Y,k=1)
train.err1=mean(train.pred1!=train.Y)
confusionMatrix(train.pred1,factor.train.Y)

t5=Sys.time()
test.pred1=knn(train.X,test.X,train.Y,k=1)
t6=Sys.time()
test.err1 <- mean(test.pred1!=test.Y)
confusionMatrix(test.pred1,factor.test.Y)
```