Project 2 Interim Report #1

Group 1: Finn Womack, Hisham Jashami, and Rama Krishna Baisetti

May 2019

Introduction

The Aerial Cactus Identification data were generated to help assessing the impact of climate change. It contains a very large number of 32 x 32 thumbnail images. They are processed to have a uniform size. These images can have cactus or not. As a team, each one of us have registered in the competition website that was provided so that we will be able to download the data and have an overview about the purpose of predicting them.

Objective

The goal of this report is to predict a best model to classify whether the image has a cactus or not. In other words, our group aims to classify these images in a way that could carry similar features in order to facilitate the task of finding a classifier which is able to predict the outcome of whether having cactus in the image or not.

Data

Data was provided to us by Dr. Sharmodeep through the Kaggle Aerial Cactus Identification Challenge. It has 17500 observations. It contains two variables; first variable is the image id, and the second variable is the binary output for whether this image has a cactus or not. The binary system was coded as 1 if it has cactus and 0 otherwise. Each of the image ids were listed as a file name each of which corresponded to a 32x32 pixel image in the dataset. We found that the data set was imbalanced by calculating the mean of the classes (1 for cactus, 0 for not cactus) in the data set which was 0.751.

Dividing the Dataset

We randomly sampled the data into testing and traing subsets with 20% in the testing set and 80% in the training set. Sice our whole dataset was 17,500 observations this lead to 3,500 observations in the testinf set and 14,000 in the training set. We then split the training set further into training and validation sets for the purpose of tuning out classifier. We again used a 20/80 split and thus the validation set had 2,800 observations while the new traing set has 11,200 observations.

After splitting the data into subsets, as a sanity check, we checked that the class imbalance was preserved and found that in all subsets contained roughly 75% cactus classes.

Feature Extraction

For the feature analysis we used the Histogram of Oriented Gradients (HOG) as the feature representation method. HOG is a feature descriptor (representation of an image extracting useful information and removing unnecessary information) used in traditional computer vision and image processing for the purpose of object detection. Based on the technique we are counting the occurences of gradient orientation in localized portions of the image. From the HOG function of OpenImageR package we are creating a vector of HOG descriptors for each image which is used in our classification model. The input image is a rgb image with the dimensions of 32 * 32 * 3, We chose the HOG function with 4 cells and 6 orientations and the output feature vector is of size 96.

Classification Methodology

Tuning

For this report we tried usin both support vector machines (SVMs) and random forests (RF) all of which were trained on the 11,200 observations in the training set. First we tuned the Random forest by adjusting the mtry parameter which controls the number of features it copairs at each split. We looked at values between 5 and 15 and found that 14 had the best error rate on the validation set. (See Code Section 4) We then looked at 4 different SVM kernels: Linear, Polynomial, Sigmoid, and Radial. (See Code Section 5) We found that the Radial kernal was the best in terms of error rate on the validation set. Lastly, we looked at the cost parameter in the svm function (See Code Section 6) and found the 5 was the best value. We first looked at the values .1, .5, 1, 5, & 10 and found 5 to be the best. We then looked at the values 3, 4, 5, 6, & 7 but found 4, 5, & 6 to be the same so we used 5 as out cost value.

Results

After tuning the classifiers we found that the randpm forest had a 10.21% error rate on the validation set while the SVM had a 7.32% error rate on the validation set. Thus, overall the SVM preformed better than the random forest. One thing to consider about these classifiers though is that they are both a bit biased, likely because of the class imbalance in the data. (75% cactus, 25% non cactus) The svm having validation set error rates of 4.34% for the cactus class and 16.57% for the non cactus class while the random forest had validation set error rates of 2.03% for the cactus class and 35.63% for the non cactus class.

Appendix: R Code

Code Section 1: Load packages & read data.

```
library(OpenImageR)
library(tidyverse)
library(e1071)
library(randomForest)


# Generate files paths
#file_dir <- dirname(rstudioapi::getSourceEditorContext()$path)
file_dir <- "/home/finn/Documents/ST\ 538/big-data-group-1/Project\ 2"
train_path <- file.path(file_dir,"g1_train")
test_path <- file.path(file_dir,"g1_test")

# get training and testing file names
train_files <- list.files(train_path)
test_files <- list.files(test_path)
```

Code Section 2: Setup Function for feature extraction

```
##########################################

# feature generation function
features <- function(file_path, file_name){
  path <- file.path(file_path, file_name)

  im <- readImage(path)
  #dim(im)

  im <- rgb_2gray(im)

  #imageShow(im)

  #intBl = resizeImage(im, width = 100, height = 100, method = 'bilinear')
  #dim(intBl)

  #im = im * 255

  hog <- HOG(im, cells = 4, orientations = 6)
  return(hog)
}
```

Code section 3: Read Classifications and Transform data

```
ans <- read_csv(file.path(file_dir,"train.csv"))

##########################################

# Setup column names
len<-length(features(train_path,train_files[1]))
```

```
cn<-character()
for(a in 1:len){
  cn <- append(cn,paste("x",as.character(a), sep = ""))
}
cn <- append(cn,"y")

# Extract features and append answer to y column
X <- matrix(NA, nrow=length(train_files), ncol=len+1)
i <- 1
for(file in train_files){
  X[i,] <- append(features(train_path,file),ans$has_cactus[ans$id==file])
  i <- i+1
}
colnames(X)=cn
# Convert to tibble
X <- as_tibble(X,colnames=cn)
X <- X %>%
  mutate(y = factor(y))
```

Code section 4: Tune mtry parameter in random forest

```
set.seed(Sys.time())
train_idx <- sample(1:nrow(X),11200)

oob_err<-double(11)
val_err<-double(11)

#mtry is no of Variables randomly chosen at each split
for(mtry in 5:15){
  rf<-randomForest(y ~ . , data = X , subset = train_idx,mtry=mtry,ntree=500)
  oob_err[mtry-4] <- rf$err.rate[500] #Error of all Trees fitted

  pred<-predict(rf,X[-train_idx,])
  val_err[mtry-4]<- with(X[-train_idx,],mean(y!=pred))
}
# mtry was best at 14
```

Code section 5: Tune kernal function and cost in svm

```
val_err <- double(4)
i <- 1
for(k_type in c("linear", "sigmoid", "polynomial", "radial")){
  svmfit <- svm(y~.,data=X, kernel=k_type, subset=train_idx)
  pred<-predict(svmfit,X[-train_idx,])
  val_err[i]<- with(X[-train_idx,],mean(y!=pred))
  i = i+1
}
# output: l:0.10892857, s:0.20535714, p:0.13785714, r:0.07785714 => Radial was best

val_err <- double(5)
i <- 1
for(k in c(3,4,5,6,7)){
  svmfit <- svm(y~.,data=X, kernel="radial", cost=k, subset=train_idx)
  pred<-predict(svmfit,X[-train_idx,])
  val_err[i]<- with(X[-train_idx,],mean(y!=pred))
  i = i+1
}
# First run w/ c(.1,.5,1,5,10) => 5 was best at 0.07321429
# Second run w/ c(3,4,5,6,7) => 4,5,6 were all equal & the same so 5 what we will go with
```

Code Section 6: compair svm to random forest

```
rf <- randomForest(y~.,data=X, ntrees=1500, mtry=14, subset=train_idx)

rf_pred<-predict(rf,X[-train_idx,])
rf_err<-with(X[-train_idx,],mean(y!=rf_pred))

svm_model <- svm(y~.,data=X, kernel="radial", cost=5, subset=train_idx)

svm_pred<-predict(svm_model,X[-train_idx,])
svm_pred_1<-predict(svm_model,filter(X[-train_idx,],y=="1"))
svm_pred_0<-predict(svm_model,filter(X[-train_idx,],y=="0"))
svm_err<- with(X[-train_idx,],mean(y!=svm_pred))
svm_err_1<- with(filter(X[-train_idx,],y=="1"),mean(y!=svm_pred_1))
svm_err_0<- with(filter(X[-train_idx,],y=="0"),mean(y!=svm_pred_0))
```