Project 2 report

Subject: Machine Learning
Name: Nguyen Van Khoi
Student ID: SE61174
Class: CS0801

Course project 2, image classification

You are given the images inside the zip file mit8-images-64x64.zip which contains 2688 images in 8 categories. The same dataset as in project 1.

In this project you have to design a classifier to give each image some tags or categories.

Please complete the project and write the report by ansIring the following questions.

Q1. Load the feature table.

Please load the image into a table. The last column of the table will be the category of the image (coast, forest, mountain, country, inside city, tall building, street, highway). Other columns will be the feature vector.

Please report the number of columns and number of rows of your table.

Please explain why you choose the feature table.

For this project, I've experimented four kind of color-based featured selection: color averaging, color histogram, spatial color averaging and spatial color histogram. Actually color averaging is 1 layer spatial color averaging and so do color histogram.

```
#recursive function for spatial color averaging (if targetlayer is 1 then it return the color averaging)
colorAveraging <- function(img, r1, c1, r2, c2, currentlayer, targetlayer){
  retVal <- c();
  r = 0.0; g = 0.0; b = 0.0;
  area = (r2 - r1 + 1) * (c2 - c1 + 1);
  for (i in r1:r2){
    for (j in c1:c2){
      r <- r + img[i, j, 1] * 255;
      g <- g + img[i, j, 2] * 255;
      b <- b + img[i, j, 3] * 255;
    }
  }
  retVal <- c(retVal, c(r / area, g / area, b / area));

  if (currentlayer < targetlayer) {
    rmid <- (r1 + r2) / 2;
    cmid <- (c1 + c2) / 2;
    retVal <- c(retVal, colorAveraging(img, r1, c1, rmid, cmid, currentlayer + 1, targetlayer));
    retVal <- c(retVal, colorAveraging(img, rmid + 1, c1, r2, cmid, currentlayer + 1, targetlayer));
    retVal <- c(retVal, colorAveraging(img, r1, cmid + 1, rmid, c2, currentlayer + 1, targetlayer));
    retVal <- c(retVal, colorAveraging(img, rmid + 1, cmid + 1, r2, c2, currentlayer + 1, targetlayer));
  }
  return(retVal);
}
#recursive function for spatial color histogram (if targetlayer is 1 then it return the color histogram)
colorHistogram <- function(img, r1, c1, r2, c2, currentlayer, targetlayer, color) {
  retVal <- c();
  h <- 0 * c(1:16);
  for (i in r1:r2) {
    for (j in c1:c2) {
      x <- floor(255 * img[i, j, color] / 16.0) + 1;
      h[x] <- h[x] + 1;
    }
  }
  retVal <- c(retVal, h);
  rmid <- (r1 + r2) / 2;
  cmid <- (c1 + c2) / 2;
  if (currentlayer < targetlayer) {
    retVal <- c(retVal, colorHistogram(img, r1, c1, rmid, cmid, currentlayer + 1, targetlayer, color));
    retVal <- c(retVal, colorHistogram(img, rmid + 1, c1, r2, cmid, currentlayer + 1, targetlayer, color));
    retVal <- c(retVal, colorHistogram(img, r1, cmid + 1, rmid, c2, currentlayer + 1, targetlayer, color));
    retVal <- c(retVal, colorHistogram(img, rmid + 1, cmid + 1, r2, c2, currentlayer + 1, targetlayer, color));
  }
  return(retVal);
}
```

```
#read the feature table, enable fromCSV to read from existing CSV, disable fromCSV to read from jpeg files
readInput <- function(fromCSV = TRUE, files){
  inputData <- rbind();
  if (fromCSV){
    inputData <- rbind("avgLayer1.csv", "avgLayer2.csv", "avgLayer3.csv", "histogramLayer1.csv",
"histogramLayer2.csv", "histogramLayer3.csv");
  } else {


    #define additional data
    for (layer in 1: 3) {
      inputData <- rbind(inputData, generateCSVData("avg", layer, files, n));
      inputData <- rbind(inputData, generateCSVData("histogram", layer, files, n));
    }

  }
  return(inputData);
}
```

All feature table have 2688 rows (number of jpeg files). Excluding the "Category" column, for each table:

- Color averaging (see avgLayer1.csv): 3 columns
- 2-layer spatial color averaging (see avgLayer2.csv): 15 columns
- 3-layer spatial color averaging (see avgLayer3.csv): 63 columns
- Color histogram (see histogramLayer1.csv): 48 columns
- 2-layer spatial color histogram (see histogramLayer2.csv): 240 columns
- 3-layer spatial color histogram (see histogramLayer3.csv): 1008 columns

The more detailed feature vector, the more time to generate them from jpeg files, build the model and classify. To save time, I use generated CSV to load feature table.

From my observation, the feature selection choice does affect heavily to training errors and testing errors if using SVM, logistic regression and boosting but it doesn't if using random forest.
You can see more details in the graph on next question.

Q2. Design a classifier to give a tag for each image.

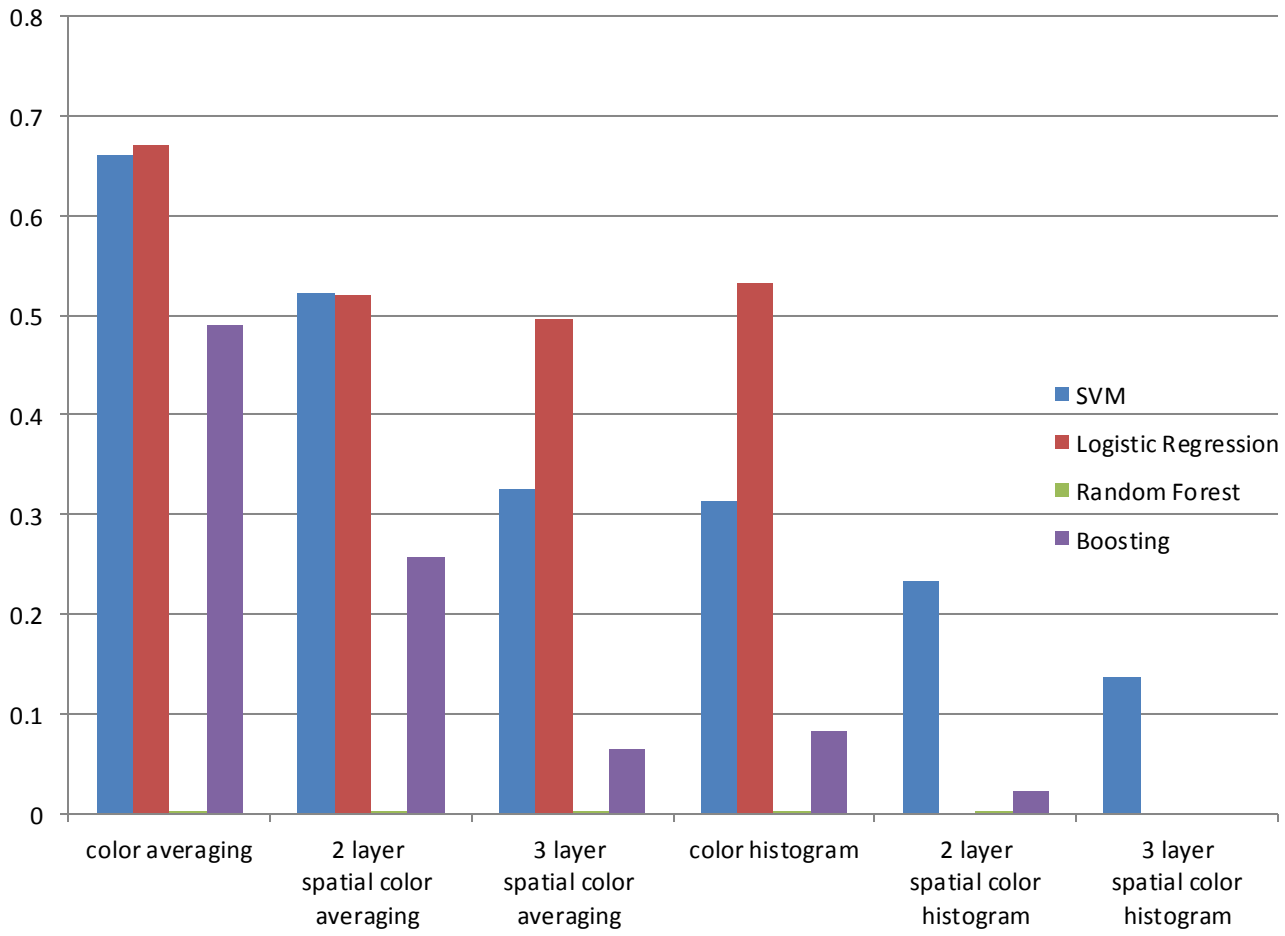
You can choose one of the classification method you have learned (svm, boosting, bagging, random forest, logistic regression).

Report the error on your training set.

NOTE:

- The original table is in Err.csv
- In SVM, I choose the degree of polynomial that generates the least training error (from degree 1 to degree 10). The chosen degree is already mentioned in README.md.
- The multinom function for logistic regression can't work with more than 1000 weights. It doesn't provide any parameter to increase this limit. As the documentation say "The maximum allowable number of weights. There is no intrinsic limit in the code, but increasing MaxNWts will probably allow fits that are very slow and time-consuming." so I skip logistic regression for 2 and 3 layers of spatial color histogram.
- The time complexity and space complexity of random forest and boosting are quite big. I've try several times on my old laptop to run them with 3-layer of spatial color histogram but the random forest take hours but not completed and boosting run out of memory so I skip.

Please refer to sheet trainingErr of Err_modified.xlsx. For a brief description please see the chart below



As you see from the chart, logistic regression seems to be the worst. Despite the details of feature, its error always goes around 50% or more. The default multinom even can't handle more than 1000 weights.

SVM is a little bit better. When feature vectors are more detailed, its errors also decrease. Moreover, SVM is the fastest. It just take over half an hour to complete the most detailed feature selection – the 3 layer spatial color histogram.

The boosting (actually it is gradient boosting) training error is better than SVM. Its errors drop significantly when there are more details. However, it consumes a huge amount of RAM and running time.

The random forest seems to be the best. Its training error is nearly perfect. But similar to boosting, it has quite big time complexity.

In short, when the feature vectors go more detailed (increase the layers or change from averaging to histogram) the training error goes lower. Of course the training time also increases depend on length of each vector. For each feature selection, the training error of each method from highest to lowest can be ordered as logistic regression, SVM, boosting, random forest; the running time and memory consume from lowest to highest can be orders as SVM, logistic regression, random forest, boosting.

Q3. Produce the output

You have to select 30% of 2688 images in the dataset as the testing set. Then you write the code to produce HTML tags for each image.

NOTE: same as previous question.

The code to generated output html file

```
outputToFile <- function(method, setName, result, expected, fileList, featureType, fileIndex = NULL) {
  imgType <- c("coast","forest","highway","insidecity","mountain","opencountry","street","tallbuilding");
  #output will be in the input folder
  e1 <- abs(result != expected);
  result <- as.data.frame(unclass(t(result)));
  expected <- as.data.frame((unclass(t(expected))));
  sink(file = paste("output", method, ".html", sep =""), append  = TRUE,  type = "output");
  cat("<style>
            .cell { display:inline-block; border: solid 1px #aeaeae; padding: 5px; margin: 3px; border-radius: 3px;
background-color: lavender; }
      .small { width:64px; }
      .tag { color: red; }
      </style>");
  cat("<h1>", setName ,"error for ",featureType, ": ", sum(e1) * 100.0/ncol(result),"%<h1>\n");


  for (j in 1 : ncol(result)) {
    for (i in 1: length(imgType)){
      if (result[j] == i) {
        cat("<div class=cell>");
        cat("<!--", colnames(result)[j], "-->");
        imgIndex <- 0;
        #my implementation make SVM produce the result set a little difference than others
        if (grepl("SVM", method)){
          imgIndex = as.numeric(colnames(result)[j]);
        } else {
          imgIndex = as.numeric(fileIndex[j]);
        }

        if (result[j] == expected[j]){
          cat ("<img class=small src = ", fileList[imgIndex], ">\n");
        } else {
          cat ("<img src = ", fileList[imgIndex], " border=5>\n");
        }
        cat("<br><span class=tag><font size=3>", imgType[i]);
        cat("</font></span></div>");
        break;
      }
    }

  }
  cat("<hr>\n");
  sink(file = NULL, type = "output");
}
```

The output html is similar to below. The name structure and its meaning are described in README.md. Those images that have thick black border are wrong tag from testing (compare to the extracted tag in file names).

Training error for avgLayer1.csv : 66.0828 %

For detailed testing error, please refer to sheet testing error in Err_modified.xlsx.

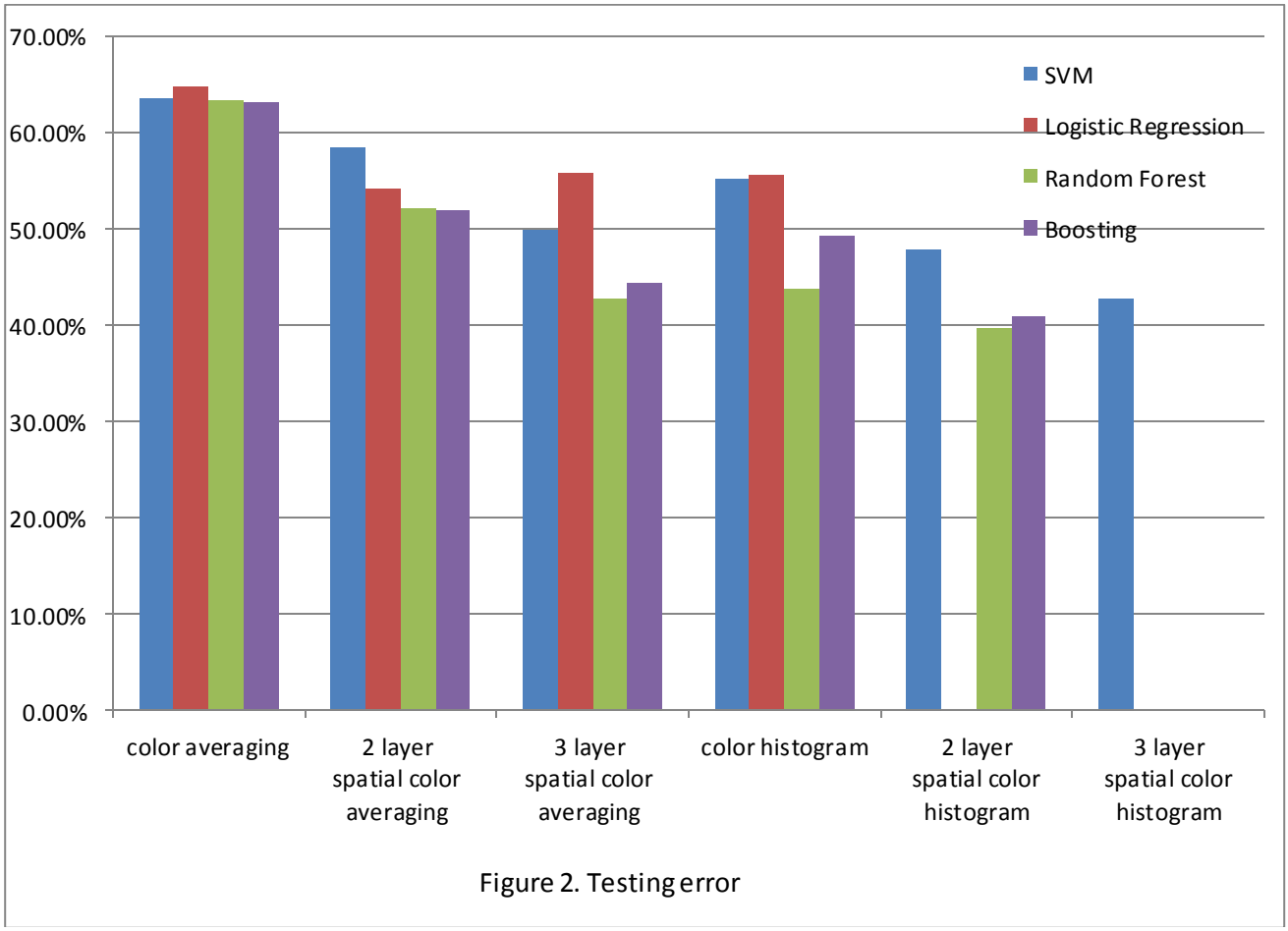


Figure 2. Testing error

For logistic regression, the training error is as expected. This method is not suitable for image classification.

For SVM, the error is quite high. It looks like some kind of overfitting. That can be my fault when loop from 1 to 10 and choose the best fit degree for training set.

For random forest and boosting, despite the lower error than others, it really surprisingly high compare to training error. Actually, the learning algorithm is overfitting. It generates unexpected differences.

In short, the testing error overall is quite high. That means the learning method or feature selection is not good. But I realize the main reasons is feature selection (you can read the "Something I found interesting" on the last page for more details).

To make result go better, you have to make feature vector more detailed. But it increases the running time and required memory, especially for random forest and boosting.

Conclusion

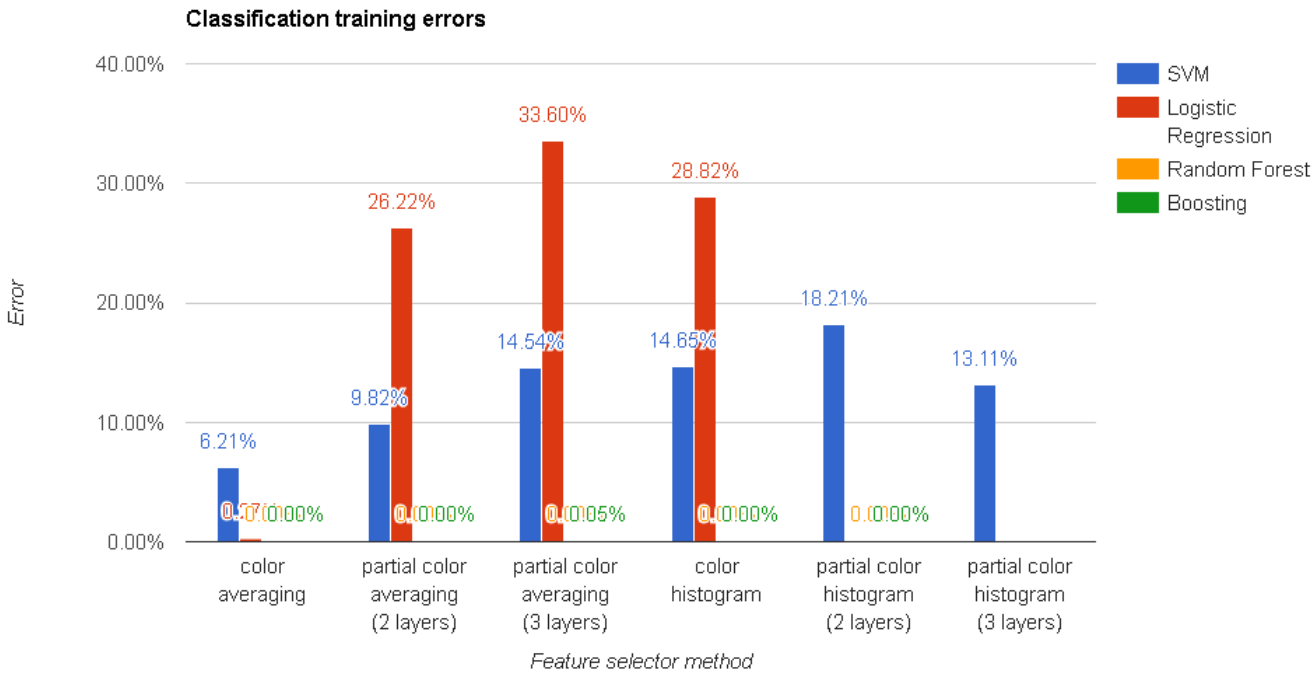
Finally, the comparison between feature selections and learning method can be briefly summary:

- Time and memory (from lowest to highest): SVM, logistic regression, random forest, gradient boosting
- Errors (from lowest to highest): gradient boosting ~ random forest, SVM, logistic regression
- Don't use logistic regression for image classification.
- The combination of simple learning method with efficient feature selection can generate acceptable result.

Something I found interesting.

These are my wrong results due to chosen wrong feature vector. Actually, when I created dataset, I add one more column that is the order reading vector from image. It's easy to see that column have linear relative to the Category.

As you see, the result (for both training set and testing set) seems to be very good comparing to the previous one. Also you can look the html in wrong folder.



The more details (more columns of each vector), the worse result of SVM and logistic regression. Random forest and boosting give the "nearly perfect" result after all.

So I conclude that, those learning method is good or not depending on its ability to "find out and use key feature".

- Logistic regression seem get lost if the key feature is mixed in a "forest of feature", the reason why it has so low errors in color averaging may be it has too few dimensions.
- SVM is a little better. At least it recognizes the linear relative between the wrong column and the Category (almost its polynomial has degree 1). However, it seem can't apply that key to classify testing set.
- The random forest and boosting are the best that can both learn and use key feature properly. However, boosting consume a lot of time and memory.

In short, we need a good feature selection that can generate key feature and a good algorithm that can learn and use these key features.

I wonder whether this can be used to get a better result using machine learning.