Classification and analysis of skin lesion images

1. Introduction

The 2019 ISIC Skin Lesion Images Classification dataset was chosen for this study. It is essentially an image classification problem, with the data consisting of 25331 labeled images from 8 different diagnostic categories, namely Melanoma, Melanocytic nevus, Basal cell carcinoma, Actinic keratosis, Benign keratosis, Dermatofibroma, Vascular lesion, and Squamous cell carcinoma. The total data size is approximately 9.2 GB. The images depict skin lesions, which are any areas of skin that differ from the surrounding skin in color, shape, size, and texture.

Figure 1: *Sample Image of labeled skin lesion data*

The ground truth histogram shown below demonstrates the data imbalance in which a few categories dominate over others. It is always advisable to test several algorithms on such data to see which one performs best. In terms of image classification, convolutional neural nets are said to perform better than other machine learning techniques because when we convert an image to a feature vector, we lose a lot of spatial information in the form of interaction between pixel intensities. CNNs, on the other hand, take this information into account when recognizing edges and thus outperform other methods.

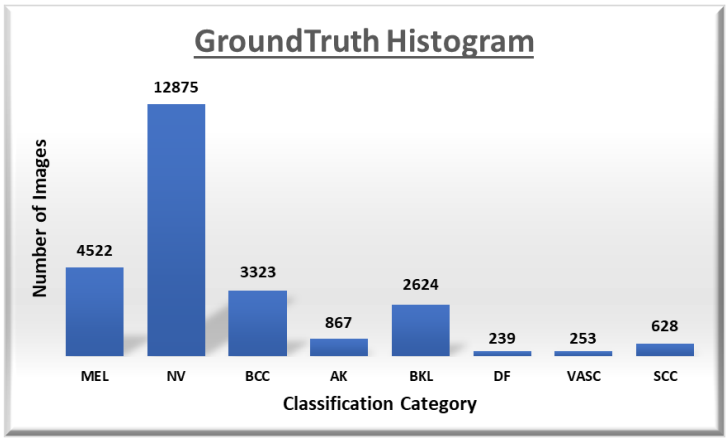


Figure 2: *Ground truth histogram of classification categories*

Nonetheless, before implementing CNN, we tried and tested other machine learning techniques, and the performance and results of these methods are discussed in the following sections.

2. Image Transformation

We had to transform our image data into a vector form with features before using any machine learning techniques so that the features could be fed into various algorithms for training and testing. The process we used to extract features from an image is shown in the figure below.

Figure 3: *Image transformation procedure*

The first step was to install and load the EBImage library in R, which contains all of the basic functions for image processing and analysis. Once the library was loaded, we fixed the pixel size. In this study, we used two independent approaches. In the first approach, we fixed the pixel size to be 64 x 64 x 3, resulting in a feature vector of size 12288 x 25331, whereas, in the second approach, the image size was fixed to be 34 x 48 x 3. The pixel size represents its length, and width and 3 represents the R, G, and B values. The image is then read from the dataset, resized to our specified dimensions, and saved in vector form. This process is looped to read images one at a time, resize them to the desired dimensions, and save the resized data in the final feature vector. Once all of the images have been read, the feature vector is saved locally in the system and used for all subsequent analyses.

3. Exploratory data analysis and dimension reduction

The data analysis presented in the current section and the next one deal with images of size 36x48. The original raw data downloaded from Kaggle consisted of 25,331 colored images of varying sizes. However, we observed that an aspect ratio of 4:3 was maintained in all images. Hence, all images were resized to a size that respects the aspect ratio. Of the 25,331 images, 75% of the data i.e. 18998 images were randomly selected to be used as the training set while the rest of the 6333 images were used as test set images. Each image is represented as a 36x48x3 vector of pixel intensities. From Table1, it is clear that there is a heavy **imbalance in the data** and that it is bound to affect classification accuracy.

Table 1: Number of images from each class label in the training set.

Class	DF	VASC	SCC	AK	BKL	BCC	MEL	NV
Proportion	0.0095	0.0097	0.0246	0.0350	0.1023	0.1310	0.1798	0.5080

A histogram of average red (R), blue (B), and green (G) pixel intensities corresponding to images of each class are shown in Figure 4. We observe that the intensity histograms corresponding to blue and green are all left-skewed with maximum intensities being close to 0.6. On the other hand, the red intensity histogram appears to be bimodal with the maximum intensity being close to 1. If we view image data to be sampled from the corresponding histograms in Figure 4, then the distributions from which different class images are being sampled look very similar.

Since each image corresponds to a feature vector of size 36x48x3 = 5184, the next logical step is to perform dimension reduction. Here, we apply Principal Component Analysis, a linear dimension reduction approach. Scree plots in Figure 5, show that 15 principal components explain 90% of the variation in the data while 90 principal

components are required to capture 96% of the variation in the data. Keeping the computational costs in mind, we believe that retaining 15 principal components is enough. Furthermore, retaining 15 principal components is backed by Kaiser's rule as well i.e. there are exactly 15 eigenvalues greater than or equal to 1.

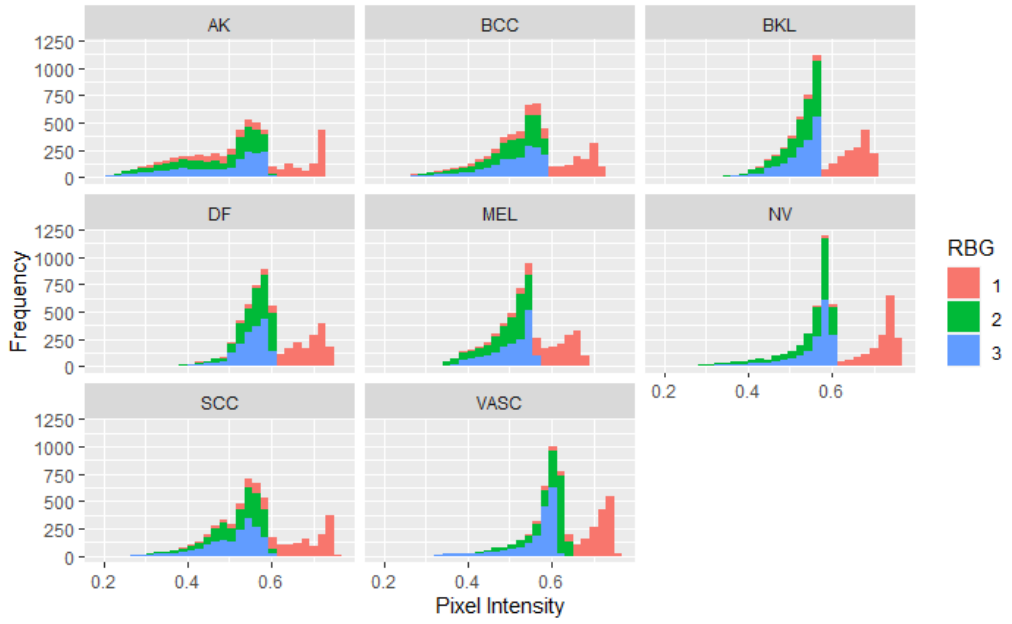
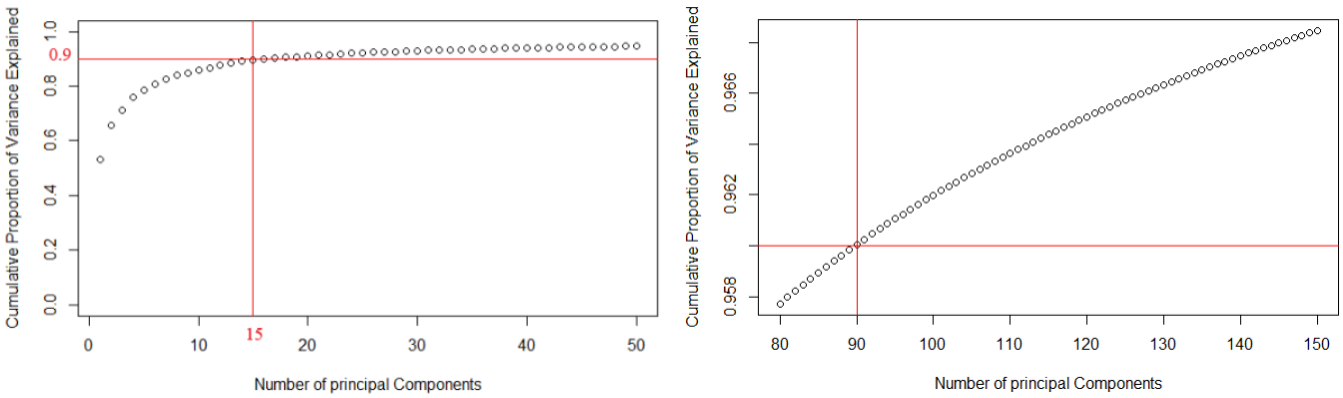


Figure 4: Histogram of RBG pixel intensities categorized by type of skin lesion.

Using these 15 principal components, we obtain a reduced feature vector of length five corresponding to each image. Different classification techniques are applied to the reduced feature set as it is to identify potential problems with the data at hand so that modifications can accordingly be made. Table 2 summarizes the results of the classification methods applied to the reduced feature set, their accuracy, and a brief comment on classification performance. Note that the data was not scaled prior to dimension reduction since all features are pixel intensities measured on the same scale.

Figure 5: Scree plots to identify the required number of PCs.



Based on our observations mentioned in Table 2, the following are the problems that must be addressed to obtain models with improved classification performance:

1. Data imbalance: This issue is dealt with by using a technique called **Synthetic Minority Over-Sampling Technique** (SMOTE). SMOTE is an over-sampling approach that over-samples the minority class by creating synthetic examples.

2. Use of information gain: Instead of bagging and boosting traditional decision trees that split based on information gain, we can focus on bagging and boosting C5.0 trees that use Gain Ratio as splitting criteria.

Table 2: Classifiers applied to the unaltered reduced feature set. The accuracy given is on the test data set.

Classifier	**Accuracy**	**Comments on classifier performance**
Single decision tree with information gain as the splitting criteria	51.5%	Due to data imbalance and the use of information gain as the splitting criteria, we observe that the trained decision tree classifies any new point into one of the three most prevalent classes - NV, MEL, and BCC
C5.0 which uses Gain ratio as the splitting criteria.	43.6%	Although lower in accuracy than the first classifier, we observe that C5.0 fits a better model than the first one in the sense that the trained tree classifies new data points into 5 different classes. This improvement is due to the use of gain ratio. However, data imbalance is still an issue.
Bagged trees with information gain as splitting criteria (500 trees)	36%	Poor performance of the bagged classifier is because the weak learners are not independent due to strong predictors like PC1. More prevalent classes like NV are classified the best while less prevalent classes are almost always misclassified. Like in the previous cases, we have the issues of data imbalance, and the use of information gain in splitting.
Random Forest involving 500 trees and 3 predictors per tree	39.5%	The performance of RF classifier is slightly better than the bagged classifier because RF decorrelates the tree topologies. However, the performance is still not high enough and very similar to the bagged trees due to the issues of data imbalance and the use of information gain.
Boosting with multiclass cross-entropy loss and 500 trees	29.7%	Although it is expected that boosting would perform better than random forests, the opposite has been observed implying the possibility of potential outliers in the data that have to be dealt with first. Like in earlier cases, data imbalance and use of information gain are still issues.
SVM with linear kernel and cost 10	51%	Linear kernel was chosen not only for its low computation cost but also because it performed better than SVMs with the radial kernel. All test set points are classified as either NV or MEL, the two most prevalent classes. The only issue here is the data imbalance.

4. Dealing with data imbalance

Upon applying SMOTE to the current training data, we observe that DF (the least prevalent class) is oversampled while NV (the most prevalent class) is downsampled. The total number of samples we have after applying SMOTE technique is 37,460. The proportion of images of each class after applying SMOTE technique is given in Table 3.

Table 3: Proportion of images in different classes after applying SMOTE over-sampling technique

Class	AK	BCC	BKL	DF	MEL	NV	SCC	VASC
Prop. (original)	0.0350	0.1310	0.1023	0.0095	0.1792	0.5080	0.0246	0.0097
Prop. (SMOTE)	0.0178	0.0664	0.0519	0.4977	0.0912	0.2576	0.0125	0.0049

Although the issues of data imbalance and use of information gain have been addressed, we haven't yet dealt with outliers. Therefore, boosted C5.0 trees are likely to perform poorly in comparison to bagged C5.0 trees if we continue to use multi-class cross-entropy. An attempt was made to look for R packages that implement boosting

using the multi-class Huber loss, however, we weren't able to find one. Even if we bag C5.0 trees using synthetic data, the problem of having strong predictors like PC1 is not lost.

Therefore, the best way ahead is to consider an RF classifier made using C5.0 trees. Or one can use an SVM as well since the only issue identified in Table 2 regarding SVM was data imbalance which is solved by SMOTE technique. A summary of the performance of classifiers applied to SMOTE synthetic data is given in Table 4.

Table 4: Classifiers applied to data generated by using the SMOTE over-sampling technique. Here we replaced decision trees with C5.0.

Classifier	**Accuracy**	**Comments on classifier performance**
Boosted C5.0 with 5 boosting iterations	39%	Boosting performance with as many as 500 trees was 29% when we used a regular decision tree. However, boosted C5.0, with just 5 boosting iterations produced an accuracy of 39%.
Bagged C5.0 with 10 trees	43%	Bagging performance improved from 36% to 43%. Also, note that we used 500 trees earlier (Table 2) while we attained higher accuracy with just 10 trees. Bagging with 100 C5.0 trees further increases the accuracy to 46.8%.

5. Drawbacks of current data analysis

In the data analysis that we did so far, we did not attempt to deal with outliers in the data which are clearly present based on the boosting results we obtained, and also the PC score plot obtained. (PC score plot is not included in the report due to lack of space.) Even if all the issues identified are somehow rectified, there are some inherent drawbacks of this approach that cannot be rectified.

1. Representing images as a vector leads to a loss of spatial information that is present in the 3D array representation of an image.
2. PCA is restricted to looking at only linear combinations of our features. Therefore, it wouldn't be as powerful as a non-linear dimension reduction approach. (Eg. Extracting features using a neural network)

6. Results for 64x64x3 Feature Vector

As mentioned earlier in the report we followed two independent approaches to test different machine learning algorithms. This section briefly discuss the results obtained by using a feature vector of size 64x64x3.

SVM:

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  300

Number of Support Vectors:  636

 ( 128 113 277 76 17 12 6 7 )


Number of Classes:  8

Levels:
 AK BCC BKL DF MEL NV SCC VASC
```

```
                Reference
Prediction  AK BCC BKL  DF MEL  NV SCC VASC
      AK     0   3   1   0   1   2   0    0
      BCC    1  15   7   0   2   5   0    1
      BKL    0   9   5   0   5  12   0    0
      DF     0   1   1   0   0   0   0    0
      MEL    0  10   2   0  15  21   0    0
      NV     0  11   2   0   6 104   0    0
      SCC    0   5   0   1   0   2   0    0
      VASC   0   0   0   0   0   0   0    0

Overall Statistics

                  Accuracy : 0.556
                    95% CI : (0.4921, 0.6186)
    No Information Rate : 0.584
    P-Value [Acc > NIR] : 0.8321
```

- *Training Accuracy: 92.7%*
- *Runtime: 1.33 mins*
- *Testing Accuracy: 55.6%*

Random Forest:

```
                Length Class  Mode
call               3   -none- call
type               1   -none- character
predicted        750   factor numeric
err.rate        4500   -none- numeric
confusion         72   -none- numeric
votes           6000   matrix numeric
oob.times        750   -none- numeric
classes            8   -none- character
importance     12288   -none- numeric
importanceSD       0   -none- NULL
localImportance    0   -none- NULL
proximity          0   -none- NULL
ntree              1   -none- numeric
mtry               1   -none- numeric
forest            14   -none- list
y                750   factor numeric
test               0   -none- NULL
inbag              0   -none- NULL
terms              3   terms  call
```

```
          Reference
Prediction  AK BCC BKL  DF MEL  NV SCC VASC
      AK     0   4   0   0   0   3   0    0
      BCC    0  18   0   0   1  12   0    0
      BKL    0   7   1   0   3  20   0    0
      DF     0   1   0   0   0   1   0    0
      MEL    0   7   0   0   8  33   0    0
      NV     0   4   0   0   3 116   0    0
      SCC    0   7   0   0   0   1   0    0
      VASC   0   0   0   0   0   0   0    0

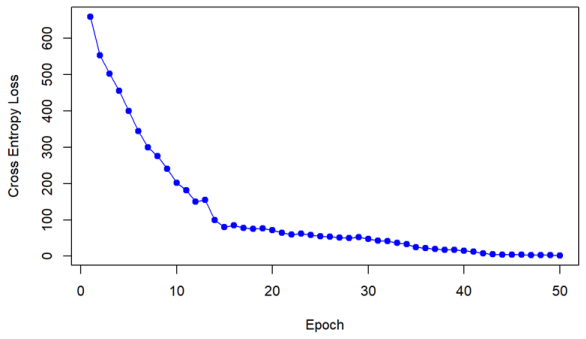
Overall Statistics

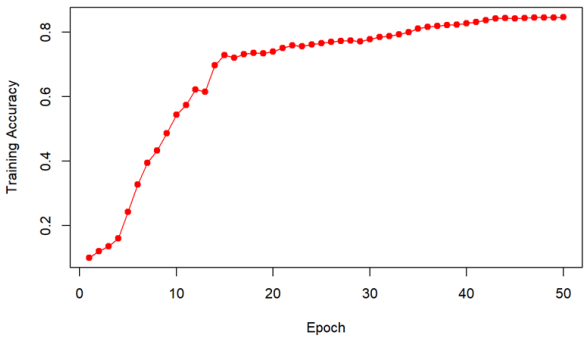
               Accuracy : 0.572
                 95% CI : (0.5081, 0.6342)
    No Information Rate : 0.744
    P-Value [Acc > NIR] : 1
```

- *Training Accuracy: 100%*
- *Runtime: 7.9 mins*
- *Testing Accuracy: 57.2%*

7. Convolutional Neural Network (CNN)

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d_11 (Conv2D)           (None, 64, 64, 32)        2432
 conv2d_10 (Conv2D)           (None, 64, 64, 32)        9248
 max_pooling2d_9 (MaxPooling2D)  (None, 32, 32, 32)     0
 conv2d_9 (Conv2D)            (None, 32, 32, 64)        18496
 max_pooling2d_8 (MaxPooling2D)  (None, 16, 16, 64)     0
 conv2d_8 (Conv2D)            (None, 16, 16, 128)       73856
 max_pooling2d_7 (MaxPooling2D)  (None, 8, 8, 128)      0
 conv2d_7 (Conv2D)            (None, 8, 8, 256)         295168
 max_pooling2d_6 (MaxPooling2D)  (None, 4, 4, 256)      0
 flatten_3 (Flatten)          (None, 4096)              0
 dense_3 (Dense)              (None, 64)                262208
 Output (Dense)               (None, 8)                 520
=================================================================
Total params: 661,928
Trainable params: 661,928
Non-trainable params: 0
_________________________________________________________________
```





- *Training Accuracy: 85%*
- *Runtime: 93.6 mins*
- *Testing Accuracy: 79.6%*

8. References

[1] Tschandl P., Rosendahl C. & Kittler H. The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161 (2018)

[2] Noel C. F. Codella, David Gutman, M. Emre Celebi, Brian Helba, Michael A. Marchetti, Stephen W. Dusza, Aadi Kalloo, Konstantinos Liopyris, Nabin Mishra, Harald Kittler, Allan Halpern: "Skin Lesion Analysis Toward Melanoma Detection: A Challenge at the 2017 International Symposium on Biomedical Imaging (ISBI), Hosted by the International Skin Imaging Collaboration (ISIC)", 2017; arXiv:1710.05006.

[3] Marc Combalia, Noel C. F. Codella, Veronica Rotemberg, Brian Helba, Veronica Vilaplana, Ofer Reiter, Allan C. Halpern, Susana Puig, Josep Malvehy: "BCN20000: Dermoscopic Lesions in the Wild", 2019; arXiv:1908.02288.

Contributions (First names mentioned alphabetically)

1. Aditya: Resized the images to 64*64*3 and extracted the features of images. Implemented support vector machine (SVM), boosting, random forest (RF), and logistic regression. Implemented Convolutional Neural Network (CNN) for image classification.
2. Padma: Attempted extracting features corresponding to images. The R implementation, and report writing correspond to sections 3, 4, and 5. (Implementation of standard classifiers, SMOTE oversampling)
3. Sudhir: Data Exploration, Image transformation process in R, feature extraction and implementing R inbuilt machine learning algorithm support vector machine, logistic regression.

Project-2 R Code

November 16, 2022

The results below are generated from an R script.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
# Load the required libraries
library(EBImage)
library(readxl)
library(tidyverse)
library(caret)
library(nnet)
library (randomForest)
library(e1071)
library(gbm)
```

```{r}

# set working directory: Set this as the directory where you have your .rmd file
and the "Data" folder
Wd_path="D:/Aditya/R"
setwd(Wd_path)

# Read the excel file as data frame. Make sure to save the file as .xlsx
instead of the default .csv extension
df = read_excel("Data/ISIC_2019_Training_GroundTruth.xlsx",col_names=TRUE)
df = df[sample(nrow(df), 1000),]

```

```{r}

Types=colnames(df)[2:9]     # The types of skin Lesion (8 types),
Unknows is not considered as none of the labels corresponds to unknown
w=64     # Width of the image
h=64      # Height of the image
N_features=w*h*3
features <- data.frame(matrix(0, nrow=dim(df)[1], ncol=(N_features+1)))
row.names(features) <- df$image
colnames(features)[N_features+1] <- "type"
```
```

```
f = matrix(0, nrow = 1, ncol=N_features)    # Empty column vector to store
the unrolled pixel intensity data

# Read the images based on type of Lesion,
resize and assign to the corresponding column in the data frame
for (i in 1:length(Types)) {
   t=Types[i]      # Type of the Leasion
   lst=df$image[df[t]==1]     # Indexes of all images belonging to that leasion type

   # Read all images of a particular type
   for (j in 1:length(lst)){
        Name_I=lst[j]  # Name of the image
        pth=paste("Data/",t,"/",Name_I,".jpg",sep="")
        #Path where image belong based on the name and type
        I = readImage(pth)      # Reading the image
        I_resized = resize(I,w,h)     # Resize into w*h
        f=t(as.vector(I_resized))        # Unroll the image.
        This unrolls column wise, picks 2nd column of red,
        places it below 1st, repets for R,G, and B
        features[Name_I,1:N_features]=f[1,1:N_features]
        features[Name_I,"type"]=t
   }

}
features$type <- as.factor(features$type)
save(features,file =paste(Wd_path,"/features.Rdata",sep=""))
```

```{r}
load(file =paste(Wd_path,"/features.Rdata",sep=""))
```

```{r}

set.seed(1)
#training.samples <- df_subset$image %>% createDataPartition(p = 0.75, list = FALSE)
# Now Selecting 75% of data as sample from total 'n' rows of the data
sample <- sample.int(n = nrow(features), size = floor(.75*nrow(features)), replace = F)
train  <- features[sample, ]
test <- features[-sample, ]
```

```{r}
# Logistic Regression
start_time <- Sys.time()
logistic <- nnet::multinom(type ~., data = train)
# Summarize the model
summary(logistic)
# Make predictions
predicted.classes <- logistic %>% predict(test)
end_time <- Sys.time()
logistic_time=end_time - start_time
head(predicted.classes)
```

```r
# Model accuracy
mean(predicted.classes == test$type)
```

```{r}
# RandomForest
start_time <- Sys.time()
RF=randomForest(as.factor(train$type)~., data = train)
# Summarize the model
summary(RF)
# Make predictions
predicted.RF.train <- RF %>% predict(train)
predicted.RF.test <- RF %>% predict(test)
end_time <- Sys.time()
RF_time=end_time - start_time
end_time - start_time
head(predicted.RF.test)
# Model accuracy
mean(predicted.RF.train == train$type)
mean(predicted.RF.test == test$type)
```

```{r}
confusionMatrix(as.factor(test$type),
                predicted.RF.test
                )

```


```{r}
#Support Vector Machine
start_time <- Sys.time()
svm = svm(type~. , data = train, kernel = "radial", cost = 300, scale = FALSE)
# Summarize the model
summary(svm)
# Make predictions
predicted.svm.train <- svm %>% predict(train)
predicted.svm.test <- svm %>% predict(test)
end_time <- Sys.time()
svm_time=end_time - start_time
end_time - start_time
head(predicted.svm.test)
# Model accuracy
mean(predicted.svm.train == train$type)
mean(predicted.svm.test == test$type)
```
```{r}
confusionMatrix(as.factor(test$type),
                predicted.svm.test
                )

```

```{r}
#Boosting
gbm_train = train
gbm_test = test
gbm_train$type=as.numeric(gbm_train$type)
gbm_test$type=as.numeric(gbm_test$type)
start_time <- Sys.time()
GBM = gbm(type~. , data = gbm_train ,distribution = "gaussian", n.trees = 500,
                  shrinkage = 0.01, interaction.depth = 4)
# Summarize the model
summary(GBM)
# Make predictions
predicted.gbm.train <- GBM %>% predict(gbm_train)
predicted.gbm.test <- GBM %>% predict(gbm_test)
end_time <- Sys.time()
boosting_time=end_time - start_time
end_time - start_time
head(predicted.gbm.train)
# Model accuracy
mean(round(predicted.gbm.train,digits = 0) == gbm_train$type)
mean(round(predicted.gbm.test,digits = 0) == gbm_test$type)

```

```{r}
decode <- function(x){
  case_when(x == 1 ~ "AK",
            x == 2 ~ "BCC",
            x == 3 ~ "BKL",
            x == 4 ~ "DF",
            x == 5 ~ "MEL",
            x == 6 ~ "NV",
            x == 7 ~ "SCC",
            x == 8 ~ "VASC",
            )
}

confusionMatrix(as.factor(sapply(gbm_test$type, decode)),
                as.factor(sapply(round(predicted.gbm.test,digits = 0), decode)) )
```

```{r}

```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()

## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
```

```
##
## locale:
## [1] LC_COLLATE=English_United States.utf8  LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9      lattice_0.20-45 png_0.1-7       digest_0.6.30   grid_4.2.2
##  [6] jsonlite_1.8.3  magrittr_2.0.3  evaluate_0.18   highr_0.9       stringi_1.7.8
## [11] rlang_1.0.6     cli_3.4.1       rstudioapi_0.14 Matrix_1.5-1    reticulate_1.26
## [16] rmarkdown_2.18  tools_4.2.2     stringr_1.4.1   tinytex_0.42    yaml_2.3.6
## [21] xfun_0.34       fastmap_1.1.0   compiler_4.2.2  htmltools_0.5.3 nnet_7.3-18
## [26] knitr_1.40
```

```
Sys.time()
```

```
## [1] "2022-11-17 04:02:17 IST"
```

Convolutional Neural Network

November 16, 2022

The results below are generated from an R script.

```
```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
# set working directory: Set this as the directory where you have your .rmd
file and the "Data" folder
Wd_path="D:/Aditya/R"
setwd(Wd_path)
```


```{r}
# Load the required libraries
library(EBImage)
library(readxl)
library(tidyverse)
library(caret)
options(scipen = 999)
library(magrittr) # needs to be run every time you start R and want to use %>%
library(dplyr)    # alternatively, this also loads %>%
library(imager)
library(keras)
```

```{r}
# Use python in your anaconda3 environment folder
reticulate::use_python("C:/Users/admin/anaconda3/envs/tf_image", required = T)
```


```{r}

folder_list <- list.files("Data_1/train/")
folder_path <- paste0("Data_1/train/", folder_list, "/")
folder_path
# Get file name
file_name <- map(folder_path, function(x) paste0(x, list.files(x))) %>% unlist()

# first 6 file name
head(file_name)
```
```

```r
# last 6 file name
tail(file_name)
length(file_name)

```
```{r}
sample_image <- sample(file_name, 6)
# Load image into R
img <- map(sample_image, load.image)

# Plot image
par(mfrow = c(2, 3)) # Create 2 x 3 image grid
map(img, plot)
```

```{r}
# Full Image Description
img <- load.image(file_name[1])
img
# Image Dimension
dim(img)

# Function for acquiring width and height of an image
get_dim <- function(x){
  img <- load.image(x)

  df_img <- data.frame(height = height(img),
                       width = width(img),
                       filename = x
                       )

  return(df_img)
}

get_dim(file_name[1])
```
```{r}

set.seed(1)
sample_file <- sample(file_name, 800)

# Run the get_dim() function for each image
file_dim <- map_df(sample_file, get_dim)

head(file_dim, 10)
summary(file_dim)
```
```{r}
# Desired height and width of images
target_size <- c(64, 64)

# Batch size for training the model
batch_size <- 32
```

```r
#library(keras)
#install_tensorflow()
# Image Generator
train_data_gen <- image_data_generator(validation_split = 0.25)
# Training Dataset
train_image_array_gen <- flow_images_from_directory

            (directory = "Data_1/train/", # Folder of the data
            target_size = target_size, # target of the image dimension (64 x 64
            color_mode = "rgb", # use RGB color
            batch_size = batch_size ,
            seed = 1,  # set random seed
            subset = "training", # declare that this is for training data
            generator = train_data_gen
            )

# Validation Dataset
val_image_array_gen <- flow_images_from_directory

            (directory = "Data_1/train/",
            target_size = target_size,
            color_mode = "rgb",
            batch_size = batch_size ,
            seed = 1,
            subset = "validation", # declare that this is the validation data
            generator = train_data_gen
            )
```

```r
# Number of training samples
train_samples <- train_image_array_gen$n

# Number of validation samples
valid_samples <- val_image_array_gen$n

# Number of target classes/categories
output_n <- n_distinct(train_image_array_gen$classes)

# Get the class proportion
table("\nFrequency" = factor(train_image_array_gen$classes)
      ) %>%
  prop.table()
```

```r

# Set Initial Random Weight
tensorflow::tf$random$set_seed(1)
model <- keras_model_sequential(name = "simple_model") %>%

  # Convolution Layer
```

```
  layer_conv_2d(filters = 16,
                kernel_size = c(3,3),
                padding = "same",
                activation = "relu",
                input_shape = c(target_size, 3)
                ) %>%

  # Max Pooling Layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Flattening Layer
  layer_flatten() %>%

  # Dense Layer
  layer_dense(units = 16,
              activation = "relu") %>%

  # Output Layer
  layer_dense(units = output_n,
              activation = "softmax",
              name = "Output")

model
```

```{r}
model %>%
  compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(lr = 0.01),
    metrics = "accuracy"
  )

# Fit data into model
history <- model %>%
  fit(
  # training data
  train_image_array_gen,

  # training epochs
  steps_per_epoch = as.integer(train_samples / batch_size),
  epochs = 30,

  # validation data
  validation_data = val_image_array_gen,
  validation_steps = as.integer(valid_samples / batch_size)
)

plot(history)
```
```{r}
val_data <- data.frame(file_name = paste0("Data_1/train/", val_image_array_gen$filenames))
%>%  mutate(class = str_extract(file_name, "AK|BCC|BKL|DF|MEL|NV|SCC|VASC"))
```

```
head(val_data, 10)
```
```{r}
# Function to convert image to array
image_prep <- function(x) {
  arrays <- lapply(x, function(path) {
    img <- image_load(path, target_size = target_size,
                      grayscale = F # Set FALSE if image is RGB
                      )

    x <- image_to_array(img)
    x <- array_reshape(x, c(1, dim(x)))

  })
  do.call(abind::abind, c(arrays, list(along = 1)))
}
```

```{r}
test_x <- image_prep(val_data$file_name)

# Check dimension of testing data set
dim(test_x)
```

```{r}
pred_test <- model %>% predict(test_x) %>% k_argmax()
head(pred_test, 10)
# Convert encoding to label
decode <- function(x){
  case_when(x == 0 ~ "AK",
            x == 1 ~ "BCC",
            x == 2 ~ "BKL",
            x == 3 ~ "DF",
            x == 4 ~ "MEL",
            x == 5 ~ "NV",
            x == 6 ~ "SCC",
            x == 7 ~ "VASC",
            )
}

pred_test <- sapply(pred_test, decode)

head(pred_test, 10)
```


```{r}
confusionMatrix(as.factor(pred_test),
                as.factor(val_data$class)
                )
```

```{r}
model_big <- keras_model_sequential() %>%

  # First convolutional layer
  layer_conv_2d(filters = 32,
                kernel_size = c(5,5), # 5 x 5 filters
                padding = "same",
                activation = "relu",
                input_shape = c(target_size, 3)
                ) %>%

  # Second convolutional layer
  layer_conv_2d(filters = 32,
                kernel_size = c(3,3), # 3 x 3 filters
                padding = "same",
                activation = "relu"
                ) %>%

  # Max pooling layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Third convolutional layer
  layer_conv_2d(filters = 64,
                kernel_size = c(3,3),
                padding = "same",
                activation = "relu"
                ) %>%

  # Max pooling layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Fourth convolutional layer
  layer_conv_2d(filters = 128,
                kernel_size = c(3,3),
                padding = "same",
                activation = "relu"
                ) %>%

  # Max pooling layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Fifth convolutional layer
  layer_conv_2d(filters = 256,
                kernel_size = c(3,3),
                padding = "same",
                activation = "relu"
                ) %>%

  # Max pooling layer
  layer_max_pooling_2d(pool_size = c(2,2)) %>%

  # Flattening layer
  layer_flatten() %>%
```

```
  # Dense layer
  layer_dense(units = 64,
              activation = "relu") %>%

  # Output layer
  layer_dense(name = "Output",
              units = output_n,
              activation = "softmax")

model_big
```

```{r}
model_big %>%
  compile(
    loss = "categorical_crossentropy",
    optimizer = optimizer_adam(lr = 0.001),
    metrics = "accuracy"
  )

history <- model %>%
  fit_generator(
  # training data
  train_image_array_gen,

  # epochs
  steps_per_epoch = as.integer(train_samples / batch_size),
  epochs = 50,

  # validation data
  validation_data = val_image_array_gen,
  validation_steps = as.integer(valid_samples / batch_size),

  # print progress but don't create graphic
  verbose = 1,
  view_metrics = 0
)

plot(history)
```

```{r}
pred_test <- predict_classes(model_big, test_x)

head(pred_test, 10)
# Convert encoding to label
decode <- function(x){
  case_when(x == 0 ~ "AK",
            x == 1 ~ "BCC",
            x == 2 ~ "BKL",
            x == 3 ~ "DF",
            x == 4 ~ "MEL",
            x == 5 ~ "NV",
```

```
            x == 6 ~ "SCC",
            x == 7 ~ "VASC",
            )
}

pred_test <- sapply(pred_test, decode)

head(pred_test, 10)
```


```{r}
confusionMatrix(as.factor(pred_test),
                as.factor(val_data$class)
                )
```

```{r}
knitr::stitch('myscript.r')
```

The R session information (including the OS info, R version and all packages used):

```
sessionInfo()

## R version 4.2.2 (2022-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 19044)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.utf8  LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8 LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.9      lattice_0.20-45 png_0.1-7       digest_0.6.30   grid_4.2.2
##  [6] jsonlite_1.8.3  magrittr_2.0.3  evaluate_0.18   highr_0.9       stringi_1.7.8
## [11] rlang_1.0.6     cli_3.4.1       rstudioapi_0.14 Matrix_1.5-1    reticulate_1.26
## [16] tools_4.2.2     stringr_1.4.1   tinytex_0.42    xfun_0.34       fastmap_1.1.0
## [21] compiler_4.2.2  htmltools_0.5.3 knitr_1.40

Sys.time()

## [1] "2022-11-17 03:38:11 IST"
```

````
---
title: "Project 2"
author: ''
date: "11/2/2022"
output:
  pdf_document: default
  html_document:
    df_print: paged
---



```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

```{r}
# Load the required libraries
library(EBImage)
library(readxl)
library(tidyverse)
library(caret)
library(nnet)
library (randomForest)
library(e1071)
```

```{r}

# set working directory: Set this as the directory where you have your .rmd
file and the "Data" folder
Wd_path="C:/Users/tpr16/OneDrive/Desktop/3rd sem/557 Data Mining I/project 2"
setwd(Wd_path)

# Read the excel file as data frame. Make sure to save the file as .xlsx
instead of the default .csv extension
# class labels for all images
df = read_excel("./Data/ISIC_2019_Training_GroundTruth.xlsx",col_names=TRUE)
dim(df)
head(df)

```

```{r}

# The types of skin Lesion (8 types), Unknown is not considered as none of the
labels corresponds to unknown
Types = colnames(df)[2:9]


w = 36    # Width of the image
h = 48   # Height of the image
N_features = w*h*3 # number of features
````
````

```
# matrix to store features, one image per row
features <- data.frame(matrix(0, nrow = dim(df)[1], ncol= (N_features+1) ))
row.names(features) <- df$image

# last column is the class label
colnames(features)[N_features+1] <- "type"



f = matrix(0, nrow = 1, ncol=N_features)   # Empty row vector to store the
unrolled pixel intensity data
dim(f)



# Read the images based on type of Lesion, resize and assign to the
corresponding column in the data frame
for (i in 1:length(Types)) {

   t=Types[i]     # Type of the Leasion
   lst=df$image[df[t]==1]    # Indexes of all images belonging to that leasion
type

   # Read all images of a particular type
   for (j in 1:length(lst)){
        Name_I=lst[j]  # Name of the image
        pth=paste("Data/",t,"/",Name_I,".jpg",sep="")    #Path where image
belong based on the name and type
        I = readImage(pth)     # Reading the image
        I_resized = resize(I,w,h)    # Resize into w*h
        f=t(as.vector(I_resized))       # Unroll the image. This unrolls
column wise, picks 2nd column of red, places it below 1st, repets for R,G, and
B
        features[Name_I,1:N_features]=f[1,1:N_features]
        features[Name_I,"type"]=t
   }

 }
save(features,file =paste(Wd_path,"/features.Rdata",sep=""))
```

In each row, we have 5185 features corresponding to one image.
* cells 1 to 36*48 (= 1728) for corresponding to red
* cells 1729 to 3456 corresponding to blue
* cells 3457 to 5184 corresponding to green

Last column of matrix corresponds to Lesion type
```{r}
load(file =paste(Wd_path,"/features.Rdata",sep=""))
dim(features)
features[1,1:100]

features$type = as.factor(features$type)
round(table(features$type)/sum(table(features$type)), 4)
```

```

# Divide into Testing and training sets
```{r}

test_size = round(0.75*dim(features)[1],0)
set.seed(1)
sample_points <- sample(1:nrow(features), test_size)

training.samples = features[sample_points,]
dim(training.samples)
# proportion of each type of images
sort(round(table(training.samples$type)/sum(table(training.samples$type)), 4))


testing.samples = features[-sample_points,]
dim(testing.samples)
```

EDA

Average pixel intensities by "type"
```{r}

Avg_intensities = aggregate(training.samples[,1:(dim(training.samples)[2]-1)],
list(training.samples$type), mean)
dim(Avg_intensities)

RBG = factor(c(rep(1,w*h), rep(3,w*h), rep(2,w*h)))
length(RBG)
Avg_int_transpose = t(Avg_intensities[,2:5185])
colnames(Avg_int_transpose) = as.character(Avg_intensities[,1])
data.frame(Avg_int_transpose)

# i for red, 3 for blue; 2 for green
Avg_int_transpose = data.frame(cbind(Avg_int_transpose,RBG))
dim(Avg_int_transpose)


library(data.table)
# convert data to long form
Avg_int_transpose2 = melt(Avg_int_transpose, id.vars = c("RBG"), measure.vars
= c("AK", "BCC", "BKL", "DF", "MEL", "NV", "SCC", "VASC"))
dim(Avg_int_transpose2)
head(Avg_int_transpose2)
colnames(Avg_int_transpose2)[2] = "type"


library(ggplot2)
library(dplyr)

 Avg_int_transpose2$RBG = as.factor(Avg_int_transpose2$RBG)
 Avg_int_transpose2 %>%
    ggplot(aes(value, fill = RBG)) + geom_histogram() + facet_wrap(~type) +
   xlab("Pixel Intensity") + ylab("Frequency")
```
```

```

# Dimension reduction using PCA
Summary - Based on both Kaiser's rule and examination of scree plot, we
believe that retaining 15 PCs is good enough.
```{r}

# no scaling needed since all intensities lie between 0 and 1
pr.out <- prcomp(training.samples[,-ncol(training.samples)])
save(pr.out,file =paste(Wd_path,"/pca.Rdata",sep=""))
load(file =paste(Wd_path,"/pca.Rdata",sep=""))

#prop of variance explained by each PC
pr.var <- pr.out$sdev^2
prop_var <- pr.var / sum(pr.var)
prop_var[1:15]


cumsum(prop_var)[96]

# plot of cum. prop. of variance explained
plot(cumsum(prop_var), xlab = "Principal Component",
    ylab = "Cumulative Proportion of Variance Explained",
    ylim = c(0, 1), type = "p")
abline(h=0.96, v = 90,  col = "red")


# close up: First 10 PCs explain more than 80% variance in data
plot(cumsum(prop_var)[1:10], xlab = "Principal Component",
    ylab = "Cumulative Proportion of Variance Explained", xlim = c(0,10),
    ylim = c(0, 1), type = "p")
abline(h = 0.8, col = "red")


# 15 PCs explain 90% variation
plot(cumsum(prop_var)[1:50], xlab = "Number of principal Components",
    ylab = "Cumulative Proportion of Variance Explained", xlim = c(1,50),
    ylim = c(0, 1), type = "p")
abline(h = 0.9, v = 15, col = "red")


# 90 PCs explain 96% variation
plot(c(80:150),cumsum(prop_var)[80:150], xlab = "Number of principal
Components",
    ylab = "Cumulative Proportion of Variance Explained", type = "p")
abline(h=0.96, v = 90,  col = "red")
```
```

```
# Kaiser's rule
pr.var[pr.var>=1]
length(pr.var[pr.var>=1])
```


Rotated co-ordinates (restricted to 15 PCs)
```{r}

# reference: https://hastie.su.domains/ISLR2/Labs/Rmarkdown_Notebooks/Ch12-unsup-lab.html


dim(pr.out$rotation)

fifteen_PCs = as.matrix(pr.out$rotation[,1:15])
sum(fifteen_PCs[,1]^2) # adds to 1 as expected

# reduced dimension after projecting data onto first 15 PCs
train_red_features = as.matrix(training.samples[,-ncol(training.samples)]) %*% fifteen_PCs
dim(train_red_features)

train_red_features = data.frame(train_red_features)
head(train_red_features)

save(train_red_features,file =paste(Wd_path,"/reduced_features.Rdata",sep=""))
load(file =paste(Wd_path,"/reduced_features.Rdata",sep=""))
```


Interpreting PCs

score plots - no visible clustering observed in the
```{r}
# reference:  https://www.geo.fu-berlin.de/en/v/soga/Geodata-analysis/Principal-Component-Analysis/principal-components-basics/Derive-synthetic-variables/index.html

Z = train_red_features[,1:4]
names(Z)
head(Z)

Z1 = cbind(Z,training.samples[,ncol(training.samples)])
dim(Z1)
colnames(Z1)[dim(Z1)[2]] = "type"
Z1$type = factor(Z1$type)
head(Z1)

# no visible clustering
Z1 %>%
    ggplot(aes(PC1, PC2, col = type)) + geom_point()  # + facet_wrap(~type)


#### extra ############################
```

```
Z1 %>%
    ggplot(aes(PC1, PC2)) + geom_point()


plot(Z[,1:2], xlab = 'Data projected along PC1', ylab = 'Data projected along
PC2')
abline(h = mean(Z[,1]), col = "blue")
abline(v = mean(Z[,2]), col = "green")
```

Loading plot
```{r}


```

Other PCA interpretatons, if possible
```{}

```


Dimension reduction on test data
```{r}
pr.out.test = prcomp(testing.samples[,-ncol(testing.samples)])
save(pr.out.test,file = paste(Wd_path,"/test_reduced.Rdata", sep="" ))
load(file = paste(Wd_path,"/test_reduced.Rdata", sep="" ))


Z2 = pr.out.test$rotation # matrix of PCs
fifteen_PCs_test = as.matrix(Z2[,1:15])
dim(Z2)
sum(fifteen_PCs_test[,1]^2) # sum is one as expected


test_red_features = as.matrix(testing.samples[,-ncol(testing.samples)]) %*%
fifteen_PCs_test
dim(test_red_features)


class(testing.samples[,ncol(testing.samples)])
test_labels = testing.samples[,ncol(testing.samples)]
test_labels = factor(test_labels)
head(test_labels)
```

Tree based methods

Single tree:
Classifies into three most prevalent categories. Problems:
1) imbalanced data
2) too many categories and we are using information gain
```{r}
```

```
# reference: https://rstudio-pubs-static.s3.amazonaws.com/
222569_a8d12e00f8204a479e84a33b49e54790.html
# https://stat.ethz.ch/R-manual/R-devel/library/rpart/html/rpart.control.html



library(rpart)
library(rpart.plot)

head(train_red_features)
type = factor(training.samples[,ncol(training.samples)])
train_red_features = cbind(train_red_features, type)
head(train_red_features)

# inbuilt rpart stopping rules are only able to classifiy into three classes
# inbuilt rpart stopping criteria  : rpart.control(minsplit = 20, minbucket =
round(minsplit/3), cp = 0.01,
      #                              maxcompete = 4, maxsurrogate = 5,
usesurrogate = 2, xval = 10,
      #                              surrogatestyle = 0, maxdepth = 30, ...)

head(train_red_features)

tree.lesion.std = rpart(train_red_features$type ~ ., train_red_features[,-
ncol(train_red_features)], method = "class", parms = list(split =
'information'))
rpart.plot(tree.lesion.std)
printcp(tree.lesion.std)

# we observe that the three most prevalent classes BCC, MEL and NV only are
the classification categories in the tree
sort(round(table(training.samples$type)/sum(table(training.samples$type)), 4))


level = levels(training.samples$type)

tree.test = predict(tree.lesion.std, data.frame(test_red_features))

tree.pred = matrix(0, nrow = dim(tree.test)[1], ncol = 1)

for (j in 1:dim(tree.test)[1]) {

  index = which.max(tree.test[j,])
   tree.pred[j,1] = level[index]
}

tree.pred= factor(tree.pred)

# 51.5%
conf.matrix.tree = table(tree.pred, test_labels)
conf.matrix.tree
(239+3025)/dim(tree.test)[1]


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

```
# cp is the complexity parameter C_alpha(T) = C(T) + alpha* |T|
# Small α results in larger trees and potential overfitting
another_tree = rpart(training.samples$type ~ ., train_red_features,control =
rpart.control(cp = 0.002), method = "class", parms = list(split =
'information'))
bestcp <-
tree.lesion.std$cptable[which.min(tree.lesion.std$cptable[,"xerror"]),"CP"]

tree.pruned <- prune(tree.lesion.std, cp = bestcp)
rpart.plot(tree.pruned)


test_red_features
tree.lesion = rpart(training.samples$type ~ ., train_red_features, method =
"class", control = rpart.control(minsplit = 10, cp = 0.002))
rpart.plot(tree.lesion)
tree.lesion




```


```
Tree using information gain - C5.0
The model fit is called C5.0 (which is an upgrade of C4.5)
This model solves the problem of too many categories to some extent - has 6
categories, misses two less prevelant
Data still imbalanced - so low accuracy
```{r}
library(C50)
library(printr)

# https://rpubs.com/kjmazidi/195428


# someimproved - classified into 6 classes
# categories not present in tree - DF, SCC (least, third from last in amount
of samples)
C5.0_tree_train = C5.0(train_red_features$type~., data = train_red_features )
summary(C5.0_tree_train)
plot(C5.0_tree_train)
text()




### Applying on test set
class(test_red_features)
test_red_features = data.frame(test_red_features)
head(test_red_features)

test_pred_C5.0 <- predict(object=C5.0_tree_train, newdata=test_red_features,
type="class")
# plot(C5.0_tree_train) # runs for too long
```
```

```
conf.matrix = table(test_pred_C5.0, test_labels)
conf.matrix[1,1]

sum = 0
for (t in 1:8) {

  sum = sum + conf.matrix[t,t]
}

sum


# we observe only 43% classification accuracy (Since data imbalance not
addressed yet)
sum/dim(test_red_features)[1]



```

Bagging
Since PCs are uncorrelated, the trees fit for different bootstrap samples do not exhibit excessive variation. But weak learners are not independent as we have "strong predictors"

We observe 63\% classification accuracy. The most well classified is the class NV (which is most in proportion). Classification error is closest to 1 in case of classes that are less in proportion. Under representation of less prop classes in bootstrap samples.

Most classified as the two major categories - problem: imbalanced data
```{r}
set.seed(1)
bag.train.tree = randomForest(train_red_features$type~., data =
train_red_features, mtry = 15, importance = T)
bag.train.tree$

bag.conf = bag.train.tree$confusion
sort(bag.conf[,9]) # classification error for each class



# Applying on test set

bag.test = predict(bag.train.tree, test_red_features)

conf.matrix.bag = table(bag.test, test_labels)
conf.matrix.bag

sum.bag = 0
for (t in 1:8) {

  sum.bag = sum.bag + conf.matrix.bag[t,t]
}
```

```
# 36% accuracy
sum.bag/dim(test_red_features)[1]
```


Random forest classifier
Clearly, PC1 is a more important variable. Therefore bagging would not involve independent trees. RF classifier will help de-correlate the tree topologies.

Most classified into two major categories: problem = imbalanced data
```{r}
rf.train = randomForest(train_red_features$type~., data = train_red_features, importance = TRUE)
rf.train

# applying on test data
rf.test = predict(rf.train, test_red_features)

conf.matrix.rf = table(rf.test, test_labels)
conf.matrix.rf

sum.rf = 0
for (t in 1:8) {

  sum.rf = sum.rf + conf.matrix.rf[t,t]
}

# 39% accuracy
sum.rf/dim(test_red_features)[1]
```

Boosting

Is there a multi class huber loss?
```{r}
library(caret)
library(gbm)

# ref: https://rpubs.com/nkrohrmann/predictionclasse


class(train_red_features$type)
boost.train = gbm(train_red_features$type~., data = train_red_features, distribution = "multinomial", n.trees = 500)
summary(boost.train)

boost.train.5 = gbm(train_red_features$type~., data = train_red_features, distribution = "multinomial", n.trees = 5)

# apply on test data



# I used the for loop below to determine the most likely classe respectively and store it in a new data frame
```

```
boost.test <- as.data.frame(boost.test)
boost.test.5 <- as.data.frame(boost.test.5)
dim(boost.test)

test_red_features = data.frame(test_red_features)
#  the prediction it produced was a data frame that contained the likelihood
of each level of the factor variable classe for every entry.
boost.test = predict(boost.train, test_red_features)
boost.test.5 = predict(boost.train.5, test_red_features)


level = levels(training.samples$type)

boost.pred.5 = matrix(0, nrow = dim(boost.test.5)[1], ncol = 1)
dim(boost.test.5)[1] # somehow 3D array with third co-ord = 1


for (j in 1:dim(boost.test.5)[1]) {

  index = which.max(boost.test.5[j,,1])
   boost.pred.5[j,1] = level[index]
}

boost.pred= factor(boost.pred.5)

conf.matrix.boost = table(boost.pred, test_labels)
conf.matrix.boost


# 48% for 5 trees boosting
(2652+427)/dim(boost.test.5)[1]

# 29% for 200 trees
 (1034+784)/dim(boost.test)[1]


# 26% accuracy for 100 trees
(1071+563)/dim(boost.test)[1]

# 500 trees
 (1082+ 401)/dim(boost.test)[1]
```
Possible improvement: starting weight is a prior density that incorporates
imbalance in data

SVM
```{r}

svm.train = svm(train_red_features$type ~ ., data = train_red_features, kernel
= "linear", gamma = 0.1, cost = 10)

svm.test = predict(svm.train, test_red_features)

conf.matrix.svm = table(svm.test, test_labels)
```

```
(69+3162)/dim(test_red_features)[1]



svm.train = svm(train_red_features$type ~ ., data = train_red_features, kernel
= "radial", gamma = 5, cost = 10)

svm.test = predict(svm.train, test_red_features)

conf.matrix.svm = table(svm.test, test_labels)

(69+3162)/dim(test_red_features)[1]




summary(svm.train)
svm.train$decision.values
svm.test = as.factor(svm.test)


set.seed(1)
tune.out <- tune(svm, train_red_features$type ~ ., data = train_red_features,
    kernel = "linear",
    ranges = list(
      cost = c(0.1, 1, 10, 20, 100),
      gamma = c(0.5, 1, 2, 3, 4)
    )
  )
summary(tune.out)

```

Dealing with imbalenced data: Downsampling with SMOTE
```{r}
# reference: https://topepo.github.io/caret/subsampling-for-class-
imbalances.html
# https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/
trainControl

library(smotefamily)

smote_train = SMOTE(train_red_features[,-ncol(train_red_features)],
train_red_features$type)
dim(smote_train$data)
smote_syn_data = smote_train$data


names(smote_syn_data)[ncol(smote_syn_data)] = "type"
smote_syn_data$type = factor(smote_syn_data$type)

save(smote_syn_data,file = paste(Wd_path,"/smote_train.Rdata", sep="" ))
load(file = paste(Wd_path,"/smote_train.Rdata", sep="" ))

smote_prop_train = round(prop.table(table(smote_syn_data$type)),4)
```

```
orig_prop_train = round(table(training.samples$type)/
sum(table(training.samples$type)), 4)

orig_prop_train; smote_prop_train

```

```
# Applying classification algorithms on synthetic data
boosted C5.0 - if too many bagging iterations then performance gets worse due
to too many outliers.
bagged SVM - couldn't do
svm on smote data
bagged C5.0
RF on C.5
logistic
```{r}
library(caret)
library(C50)
library(kernlab)


# smote_C5.0 = train(smote_syn_data$type~., data = smote_syn_data, method =
"C5.0")
# smote_rf_train = train(smote_syn_data$type~., data = smote_syn_data, method
= "rf", trControl = ctrl)

bagctrl = bagControl(fit = svmBag$fit, predict = svmBag$pred, aggregate =
svmBag$aggregate)
ctrl = trainControl(method = "cv", number = 5)


# Random Forest
smote_rf_train = randomForest(smote_syn_data$type~., data = smote_syn_data,
ntree = 100)
save(smote_rf_train,file =paste(Wd_path,"/smote_rf_100trees.Rdata",sep=""))


# C5.0
smote_C5.0_1 =  C5.0(smote_syn_data[,-ncol(smote_syn_data)],
smote_syn_data$type, trials = 1)

# Boosted C50
smote_C5.0 =  C5.0(smote_syn_data[,-ncol(smote_syn_data)],
smote_syn_data$type, trials = 5)
save(smote_C5.0,file =paste(Wd_path,"/smote_C5.0_5trials.Rdata",sep=""))


smote_C5.0_10 =  C5.0(smote_syn_data[,-ncol(smote_syn_data)],
smote_syn_data$type, trials = 10)

smote_C5.0_50 =  C5.0(smote_syn_data[,-ncol(smote_syn_data)],
smote_syn_data$type, trials = 50)


# Bagged SVM - keep getting error
```
```

```
svmbag <- train(smote_syn_data$type~., data = smote_syn_data, "bag",
bagControl = bagctrl)


# methods
smote_rf_train
smote_C5.0$boostResults
smote_C5.0_50$boostResults

smote_rf_train
smote_rf_train$forest


# svm on smote data
svm.train.smote = svm(smote_syn_data$type ~ ., data = smote_syn_data, kernel =
"linear", gamma = 0.1, cost = 10)
save(svm.train.smote,file =paste(Wd_path,"/smote_svm.Rdata",sep=""))

svm.test.smote = predict(svm.train.smote, test_red_features)

conf.matrix.svm.smote = table(svm.test.smote, test_labels)

(69+3162)/dim(smote_syn_data)[1]



svm.train = svm(train_red_features$type ~ ., data = train_red_features, kernel
= "radial", gamma = 5, cost = 10)

svm.test = predict(svm.train, test_red_features)

conf.matrix.svm = table(svm.test, test_labels)

(69+3162)/dim(test_red_features)[1]



# predictions
smote_C5.0_1_pred = predict(smote_C5.0_1, test_red_features)
sum(diag(table(smote_C5.0_1_pred, test_labels)))/length(test_labels)

smote_C5.0_pred = predict(smote_C5.0, test_red_features)
table(smote_C5.0_pred, test_labels)
sum(diag(table(smote_C5.0_pred, test_labels)))/length(test_labels) # 39%
accuracy for just 5 trees


smote_C5.0_10_pred = predict(smote_C5.0_10, test_red_features)
table(smote_C5.0_10_pred, test_labels)
sum(diag(table(smote_C5.0_10_pred, test_labels)))/length(test_labels)



# same issue of MEL, NV
smote_C5.0_50_pred = predict(smote_C5.0_50, test_red_features)
table(smote_C5.0_50_pred, test_labels)
```

```
sum(diag(table(smote_C5.0_pred, test_labels)))/length(test_labels)


smote_rf_ptrd = predict(smote_rf_train, test_red_features)
table(smote_rf_ptrd, test_labels)
```


bagging C5.0 on smote
```{r}

library(baguette)
C5.0_bag_smote = bagger(smote_syn_data[,-ncol(smote_syn_data)],
smote_syn_data$type, base_model = "C5.0", times = 100 )

class(test_labels)
length(test_labels)



C5.0_bag_smote_pred = predict(C5.0_bag_smote, test_red_features)
class(C5.0_bag_smote_pred)
colnames(C5.0_bag_smote_pred) = "type"
dim(C5.0_bag_smote_pred)

is.na(as.numeric(C5.0_bag_smote_pred[1,1]) == as.numeric(test_labels[1]) )

tot = 0

for (j in 1:dim(C5.0_bag_smote_pred)[1]) {

  if (as.numeric(C5.0_bag_smote_pred[j,1])==as.numeric(test_labels[j])){
    tot = tot + 1
  }

}

tot # 2726
tot/dim(test_red_features)[1]


```


Dimension reduction using neural networks
```{r}
view(USArrests)

```

```{r}

df_subset=df[1:6500,]
set.seed(1)
training.samples <- df_subset$type %>% createDataPartition(p = 0.75, list =
FALSE)
train.data  <- df_subset[training.samples, ]
test.data <- df_subset[-training.samples, ]
```

```{r}
# Logistic Regression
start_time <- Sys.time()
logistic <- nnet::multinom(type ~., data = train.data)
# Summarize the model
summary(logistic)
# Make predictions
predicted.classes <- logistic %>% predict(test.data)
end_time <- Sys.time()
logistic_time=end_time - start_time
head(predicted.classes)
# Model accuracy
mean(predicted.classes == test.data$type)
```

```{r}
# RandomForest
start_time <- Sys.time()
RF=randomForest(type~., data = train.data )
# Summarize the model
summary(RF)
# Make predictions
```

```
predicted.classes <- RF %>% predict(test.data)
end_time <- Sys.time()
logistic_time=end_time - start_time
head(predicted.classes)
# Model accuracy
mean(predicted.classes == test.data$type)
```

```{r}
#Support Vector Machine
start_time <- Sys.time()
svm = svm(type~. , data = train.data, kernel = "radial", cost = 10, scale =
FALSE)
# Summarize the model
summary(svm)
# Make predictions
predicted.classes <- svm %>% predict(test.data)
end_time <- Sys.time()
logistic_time=end_time - start_time
head(predicted.classes)
# Model accuracy
mean(predicted.classes == test.data$type)
```

```{r}



for (i in dim(boost.test)[1]){

  max <- max(boost.test[i,])

  if(boost.test[i,1] == max){

    de <- "AK"
    df <- rbind(df, de)
    }
  else if(boost.test[i,2] == max){

    de <- "BCC"
    df <- rbind(df, de)
    }
  else if (boost.test[i,3] == max){

    de <- "BKL"
    df <- rbind(df, de)
    }
  else if (boost.test[i,4] == max){

    de <- "DF"
    df <- rbind(df, de)
  }
  else if (boost.test[i,5] == max){

    de <- "MEL"
```

```
    df <- rbind(df, de)
  }
  else if (boost.test[i,6] == max){

    de <- "NV"
    df <- rbind(df, de)
  }
    else if (boost.test[i,7] == max){

    de <- "SCC"
    df <- rbind(df, de)
  }
  else {
    de <- "VASC"
    df <- rbind(df, de)
  }


}

```

```{r}
library(caret)
# https://topepo.github.io/caret/subsampling-for-class-imbalances.html
set.seed(3)

down_train = downSample(train_red_features, train_red_features$type)

down_train
table(train_red_features$type)
table(down_train$type)

# too few in each sample



library(smotefamily)
set.seed(4)
smote_train = SMOTE( train_red_features[,-dim(train_red_features)[2]],
train_red_features$type )
class(smote_train)


# A resulting dataset consists of original minority instances, synthetic
minority instances and original majority instances
down_train1 = smote_train$data
dim(down_train1[,-16])
dim(train_red_features[,-16])
dim(down_train[,-c(16,17)]) # one extra type column attached
# again imbalanced



round(table(training.samples$type)/sum(table(training.samples$type)), 4)
round(prop.table(table(down_train1$class)),4)
```

```
prop.table(table(down_train$type))


table(training.samples$type)
table(down_train1$class)
table(down_train$type)


library(ROSE)
set.seed(9560)
rose_train <- ROSE(train_red_features$type ~ ., data = train_red_features)
$data



```


Applying classification algorithms on down_train1
```{r}
names(down_train1)[16] = "type"
down_train1$type = factor(down_train1$type)
C5.0_down_train1 = C5.0(down_train1$type~., data = down_train1 )




test_pred_down_C5.0 <- predict(object=C5.0_down_train1,
newdata=test_red_features, type="class")
# plot(C5.0_tree_train) # runs for too long


conf.matrix.down1 = table(test_pred_down_C5.0, test_labels)
conf.matrix[1,1]

sum1 = 0
for (t in 1:8) {

  sum1 = sum1 + conf.matrix.down1[t,t]
}


```



```{r}


```