Final Project

Andrew Tiu, Kelly Wu, Yixuan (Sherry) Wu

April 30, 2020

Contents

#Data Read-in & Clean-up

##Training set

First, read in all the train files. Each `files_{name}[i]` will display a file route directed to one photo.

```
##setup code
files_buildings = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/buildings",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T, )))
files_forest = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/forest",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_glacier = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/glacier",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_mountain = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/mountain",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_sea = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/sea",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_street = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_train/street",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
```

Since our laptops are unable to handle all the photos, we decided to randomly select 200 photos from each category's training set.

```
##randomly selected 200 photos from each category (with set.seed(1)).
set.seed(1)
f_buildings = sample(files_buildings, 200)
f_forest = sample(files_forest, 200)
f_glacier = sample(files_glacier, 200)
```

```r
f_mountain = sample(files_mountain, 200)
f_sea = sample(files_sea, 200)
f_street = sample(files_street, 200)
```

```r
##combine all six files and load the image
files = c(f_buildings, f_forest, f_glacier, f_mountain, f_sea, f_street)
image_list = lapply(files, load.image)
image_list[[1]] #what the info of the first   image looks like
## Image. Width: 150 pix Height: 150 pix Depth: 1 Colour channels: 3

##get the classname from the directory
class_list = basename(dirname(files))
table(class_list)
## class_list
## buildings    forest   glacier  mountain       sea    street
##       200       200       200       200       200       200

remove(f_buildings, f_forest, f_glacier, f_mountain, f_sea, f_street,
       files_buildings, files_forest, files_glacier,
       files_mountain, files_sea, files_street, files)
```

For each `i` in `image_list[[i]]`, `image_list[[i]]` contains info of one image. To expand these info/pixels, we need to transform each of them to a vector. Use `img_matrix` to set up the empty matirx with numbers of columns to be 67500 (150*150). For each image, the value of each pixel will be stored in each of the columns.

Use `badimage` to count for all images that are not 150*150 pixels, and these images will be discarded from further analyses.

Use the `for` loop to change each list to a vector. Details in comments. Use `img_df` to create a new data frame from the matrix.

```r
#build the structure of an empty matrix
img_matrix = matrix(ncol = 67500, nrow = length(image_list))
badimage = NULL

for(i in 1:length(image_list)){
  if(length(as.vector(image_list[[i]])) == 67500) {
    img_matrix[i,] = c(as.vector(image_list[[i]]))
    #fill in each row with one observation's pixels
    #do this for each observation unless photo does not meet requirement
  }
  else{badimage = append(badimage, i)}
  #if image does not meet requirement, record which i it is
  #so when appending class to each observation,
  #will use this badimage to get rid of the class of those not met the requirement
}

img_df = as.data.frame(img_matrix[-badimage,])
```

```r
print(dim(img_df)) #dimension of the training set
## [1]  1195 67500
```

The number of rows is not exactly 1200 because there are several images that do not meet the 150*150 size criteria, so they were discarded from further analysis.

Add class assignment to the data frame

```r
#append class and remove excessive values
new_class = class_list[-badimage]
img_df$class = as.factor(new_class)

remove(img_matrix)
```

##Test Data

The procedure should be similar.

```r
#everything the same for test data
files_buildings = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/buildings",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_forest = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/forest",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_glacier = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/glacier",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_mountain = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/mountain",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_sea = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/sea",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
files_street = mixedsort(
  sort(
    list.files("~/Data Science/Project/seg_test/street",
               pattern = "*.jpg", all.files = T, full.names = T, no.. = T)))
```

```r
f_buildings = sample(files_buildings, 200)
f_forest = sample(files_forest, 200)
f_glacier = sample(files_glacier, 200)
f_mountain = sample(files_mountain, 200)
f_sea = sample(files_sea, 200)
f_street = sample(files_street, 200)
```

```r
files = c(f_buildings, f_forest, f_glacier, f_mountain, f_sea, f_street)
image_list = lapply(files, load.image)
##class
class_list = basename(dirname(files))
remove(f_buildings, f_forest, f_glacier, f_mountain, f_sea, f_street,
       files_buildings, files_forest, files_glacier,
       files_mountain, files_sea, files_street, files)
```

```r
img_matrix = matrix(ncol = 67500, nrow = length(image_list))
badimage = NULL

for(i in 1:length(image_list)){
  if(length(as.vector(image_list[[i]])) == 67500) {
    img_matrix[i,] = c(as.vector(image_list[[i]]))
    #fill in each row with one observation's pixels
    #do this for each observation unless photo does not meet requirement
  }
  else{badimage = append(badimage, i)}
  #if image does not meet requirement, record which i it is
  #so when appending class to each observation,
  #will use this badimage to get rid of the class of those not met the requirement
}

test_df = as.data.frame(img_matrix[-badimage,])

new_class = class_list[-badimage]
test_df$class = as.factor(new_class)

remove(img_matrix, image_list)
```

#PCA

##PCA on the training set

Run the `prcomp` command to construct the PCA

```r
colclass = ncol(img_df)
img_pca = prcomp(img_df[, -colclass], center = T, scale = T)
```

See how much varaince each PCs explains, so first by computing the cumulative sum of variance explained, and output the number of PCs that expalins 75, 80, and 90 percent of total variance. THe `img_cumsum` contians the variance each pc explains. Each `pc??` is the index that contains variance closet to 75, 80, or 90 percent.

```r
#get the cumulative variance explained
pr.var = img_pca$sdev^2
pve=pr.var/sum(pr.var)

img_cumsum = cumsum(pve)
img_cumsum[1:10]
```

```
##  [1] 0.2152665 0.3474630 0.4010042 0.4335586 0.4630449 0.4867607 0.5085308
##  [8] 0.5231124 0.5364468 0.5476634

pc75 = which.min(abs(img_cumsum - 0.75))
pc80 = which.min(abs(img_cumsum - 0.80))
pc90 = which.min(abs(img_cumsum - 0.90))
pc75
## [1] 99
pc80
## [1] 170
pc90
## [1] 422
```

Build the training set data frame based on PCA results

```
#build dataframe
df75 = data.frame(img_df$class, img_pca$x[,1:pc75])
colnames(df75)[1] = "class"

print(dim(df75) )#dimension on the train dataset
## [1] 1195  100
```

##On testing set

```
#project the pcs onto the test data
pcatest = predict(img_pca, newdata = test_df)
pcatest = as.data.frame(pcatest)

#format the test data
test75 = pcatest[, 1:pc75]
test75 = data.frame(test_df$class, test75)
colnames(test75)[1] = "class"
dim(test75) # dimension of the new test data
## [1] 1195  100
```

Therefore, we now have the projected data frame

#kNN

Normalize the variables

```
#normalizing function
normalize <- function(x) {
return ((as.numeric(x) - min(as.numeric(x))) / (max(as.numeric(x)) - min(as.numeric(x)))) }

#normalize the train dataset followed by the test dataset
normdf = apply(df75[, -1], 2, normalize)
normdf = data.frame(df75$class, normdf)
colnames(normdf)[1] = "class"
normdf$class = as.factor(normdf$class)
normtest = apply(test75[,-1], 2, normalize)
```

```r
normtest = data.frame(test75$class, normtest)
colnames(normtest)[1] = "class"
```

Use 10-fold CV to assess the optimal number of neighbors on the entire data (training + testing)

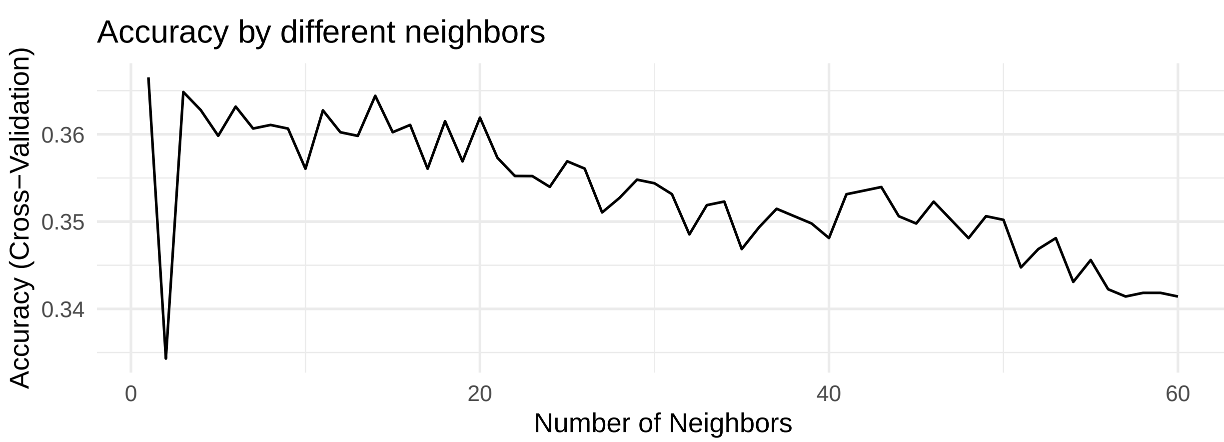
```r
#combine test and train
all75 = as.data.frame(rbind(normdf, normtest))

#use 10-fold cross-validation to find the optimal neighbors
trcontrol = trainControl(method = "cv", number = 10)
knnfit = train(class ~., method = "knn", tuneGrid = expand.grid(k = 1:60),
                trControl = trcontrol, metric = "Accuracy", data = all75)

#get the number of k  and use it in knn
kuse = knnfit$bestTune
```

The variable `kuse` is used to store the number of k that provides the highest accuracy. The following plot show the different accuracy by different number of neighbors.

```r
ggplot() +
  geom_line(aes(x = 1:60, y = knnfit$results[,2])) +
  labs(x = "Number of Neighbors", y = "Accuracy (Cross-Validation)",
       title = "Accuracy by different neighbors") +
  theme_minimal()
```



Now perform knn.

```r
knntrain = knn3Train(normdf[,-1], normtest[,-1], cl = normdf[,1], k = kuse, prob = T)
knntab = table(knntrain, normtest$class)
kable(knntab)
```

	buildings	forest	glacier	mountain	sea	street
buildings	16	0	1	1	3	15
forest	56	145	14	10	28	69
glacier	28	6	65	22	35	22
mountain	56	31	57	105	55	30
sea	31	7	41	44	61	19
street	13	11	19	16	18	45

Accuracy:

```
sum(diag(knntab))/sum(knntab)
## [1] 0.3656904
```

#Random Forest

```
hyper_grid = expand.grid(
        mtry=c(1:15),
        node_size=c(1:15),
        sampe_size=c(0.55,0.632,0.7,0.8),
        OOB_err=0
)
```

```
start.time = Sys.time()
for(i in 1:nrow(hyper_grid)) {
        ranger_rf = ranger(
                formula=class~.,
                data=df75,
                num.trees=2000,
                mtry=hyper_grid$mtry[i],
                min.node.size=hyper_grid$node_size[i],
                sample.fraction=hyper_grid$sampe_size[i],
                seed=1
        )

        hyper_grid$OOB_err[i] = ranger_rf$prediction.error
}
end.time = Sys.time()
time.taken = end.time - start.time
time.taken
## Time difference of 1.663089 hours
```

```
hyper_grid %>%
dplyr::arrange(OOB_err) %>%
head(10)
##    mtry node_size sampe_size   OOB_err
## 1    10         7      0.800 0.4276151
## 2    11        15      0.550 0.4284519
## 3     7         1      0.550 0.4292887
## 4    11         4      0.632 0.4292887
## 5    11        10      0.632 0.4309623
```

```
## 6     7         4      0.700 0.4309623
## 7     9         9      0.632 0.4317992
## 8    11        14      0.632 0.4317992
## 9     5         5      0.700 0.4317992
## 10   14         3      0.800 0.4317992
```

```
which.min(hyper_grid$OOB_err) # 775
## [1] 775

hyper_grid[775,]
##     mtry node_size sampe_size   OOB_err
## 775   10         7        0.8 0.4276151
```

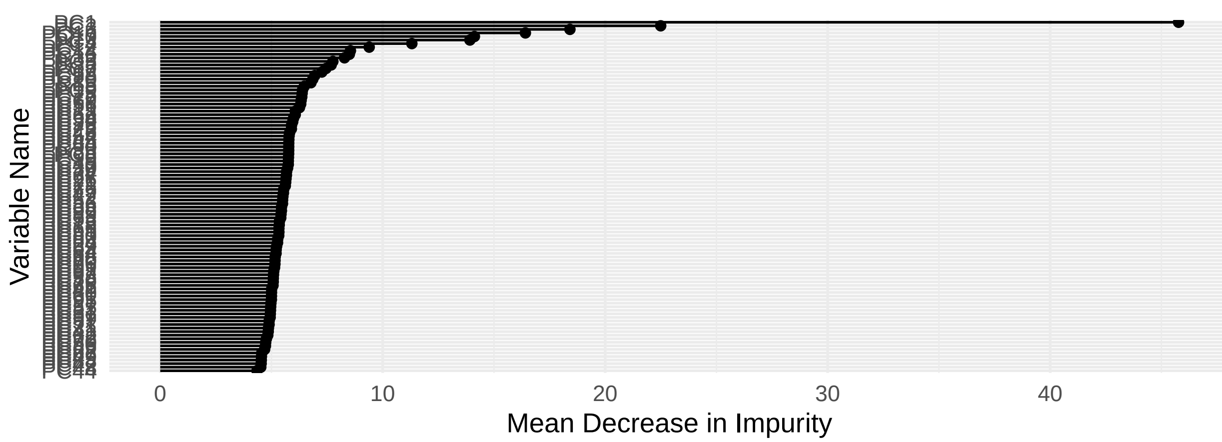
```
ranger_rf = ranger(
                formula=class~.,
                data=df75,
                num.trees=2000,
                mtry=10,
                min.node.size=7,
                sample.fraction=0.8,
                seed=1,
                importance='impurity'
        )
ranger_rf
## Ranger result
##
## Call:
##  ranger(formula = class ~ ., data = df75, num.trees = 2000, mtry = 10,      min.node.size =
##
## Type:                             Classification
## Number of trees:                  2000
## Sample size:                      1195
## Number of independent variables:  99
## Mtry:                             10
## Target node size:                 7
## Variable importance mode:         impurity
## Splitrule:                        gini
## OOB prediction error:             42.76 %
```

```
ranger_rf$confusion.matrix
##            predicted
## true         buildings forest glacier mountain sea street
##   buildings        114     19      15       12   9     31
##   forest            10    158       1        6   3     21
##   glacier           12      6     121       23  21     15
##   mountain          14     10      32      109  29      4
##   sea               18     16      50       44  59     13
##   street            29     26      13        6   3    123
```

```r
imp = ranger_rf$variable.importance
```

```r
imp = as.data.frame(imp)
head(imp)
##          imp
## PC1 45.76979
## PC2 22.49475
## PC3 13.91928
## PC4 11.31050
## PC5  7.76092
## PC6 18.41533
imp$varnames = rownames(imp)
rownames(imp) = NULL
head(imp)
##        imp varnames
## 1 45.76979      PC1
## 2 22.49475      PC2
## 3 13.91928      PC3
## 4 11.31050      PC4
## 5  7.76092      PC5
## 6 18.41533      PC6

ggplot(imp,
       aes(x=reorder(varnames, imp),
           y=imp)) +
        geom_point() +
        geom_segment(aes(x=varnames, xend=varnames, y=0, yend=imp)) +
        ylab('Mean Decrease in Impurity') +
        xlab('Variable Name') +
        coord_flip() +
        theme_minimal()
```




```r
pred_RF = predict(ranger_rf, test75)
```

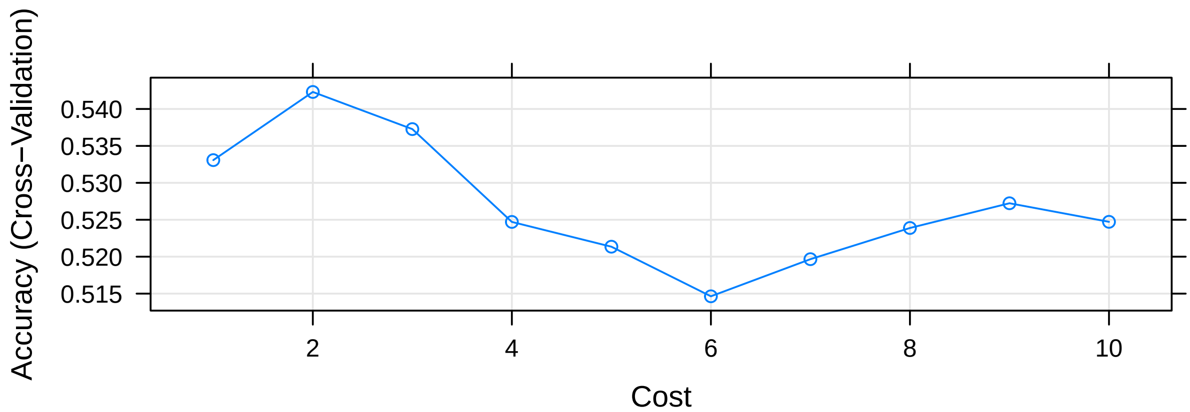
```r
table(pred_RF$predictions, test75$class)
##
##             buildings forest glacier mountain sea street
##   buildings        62      5       4        5  15     13
##   forest           26    158       7       10  20     31
##   glacier          37      6     133       32  58     34
##   mountain         25     11      19      125  39      6
##   sea              16      5      21       23  58      8
##   street           34     15      13        3  10    108
prop.table(table(pred_RF$predictions == test75$class))
##
##     FALSE      TRUE
## 0.4610879 0.5389121
```

#SVM

Use 10-fold CV to assess the optimal cost for the fitting process. Here, the radial basis kernel is chosen based on its better performance.

```r
line.svm75 <- train(x=df75[,-1], y=df75[,1],
                  method = "svmRadial",
                  trControl =trainControl(method = "cv", number = 10),
                  tuneGrid =expand.grid(C =c(1:10), sigma = 0.01))
line.svm75
## Support Vector Machines with Radial Basis Function Kernel
##
## 1195 samples
##   99 predictor
##    6 classes: 'buildings', 'forest', 'glacier', 'mountain', 'sea', 'street'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1076, 1076, 1075, 1075, 1076, 1075, ...
## Resampling results across tuning parameters:
##
##   C   Accuracy   Kappa
##    1  0.5330742  0.4397105
##    2  0.5423039  0.4507796
##    3  0.5372759  0.4447529
##    4  0.5247059  0.4296965
##    5  0.5213515  0.4256573
##    6  0.5146359  0.4175993
##    7  0.5196639  0.4236122
##    8  0.5238866  0.4286705
##    9  0.5272339  0.4326825
##   10  0.5247199  0.4296665
##
## Tuning parameter 'sigma' was held constant at a value of 0.01
```

```
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.01 and C = 2.
plot(line.svm75)
```



The final fitted model is decided using the radial basis kernel when cost is 2 and gamma is 0.01, which is confirmed in the previous CV plot.

```
# Conduct the multi-class SVM on the training set
img_svm75 <- svm(x=df75[,-1], y=df75[,1], type="C-classification",
                 kernel="radial", cost = 2, gamma = 0.01 )
summary(img_svm75)
##
## Call:
## svm.default(x = df75[, -1], y = df75[, 1], type = "C-classification",
##     kernel = "radial", gamma = 0.01, cost = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  2
##
## Number of Support Vectors:  1141
##
##  ( 200 164 192 186 200 199 )
##
##
## Number of Classes:  6
##
## Levels:
##  buildings forest glacier mountain sea street

# Prediction on test data and save the confusion matrix in svm_table
pred_75 <- predict(img_svm75, newdata=test75[,-1], type="class")
```

```
svm_table <- table(pred_75, test75$class)

# Calculate the misclassification rate
matrix=as.matrix(table(pred_75, test75$class))
diag=diag(matrix)
n=sum(matrix)
accuracy= sum(diag)/n
accuracy
## [1] 0.5305439
```

```
svm_table
##
## pred_75     buildings forest glacier mountain sea street
##   buildings        67      5      16        8  18     26
##   forest           20    150       2        4  15     24
##   glacier          19      4     115       25  40     19
##   mountain         33     14      21      118  39     12
##   sea              31     12      33       39  77     12
##   street           30     15      10        4  11    107
```