Comparison of classification models for image data with an emphasis on the importance of transforming predictors.

Diana Ojeda-Revah

Contents

1 Introduction.

This is the second capstone project required for the final course of the Data Science certification program. The goal of this project is to implement a model that best classifies data with 7 classes, making an emphasis on how some preprocessing affects the result. Three different models are implemented, making an emphasis on the importance of preprocessing procdedures and how this preprocessing affects the performance of the each model. The models implemented are knn (k nearest neighbors), classification trees and random forest. The preprocessing explored are: exclude very highly correlated variables, center-scale variables and principal components. The data is divided into test/train datasets and the algorithms, along with the choice of preprocessing compared by evaluating accuracy in the test set. The random forest algorithm is also timed, to illustrate a possible advantage of using principal components. All the models are trained with caret in the training sets, using cross-validation or bootstrap to find the tuning parameters. The test data is only used to evaluate the models.

1.1 The dataset.

The data chosen for this project can be found in https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/segment/segment.dat and is accompanied by a brief explanation here. The data consists of `2310` instances, with 19 numeric predictors, and one class response, labeled *segment*. Each of the instances is obtained from a 3 by 3 pixel region of an image corresponding to each of the classes, and from each of these 3

by 3 regions, 18 characteristics related to color, contrast, intensity and hue are obtained. Each of the images corresponds to one of these classes (this is the response variable that we are trying to predict):

- Brickface
- Sky
- Foliage
- Cement
- Window
- Path
- Grass

The numeric predictors as described in https://archive.ics.uci.edu/ml/machine-learning-databases/statlog/segment/segment.doc are:

1. reg-cnt-col: the column of the center pixel of the region.
2. reg-cnt-row: the row of the center pixel of the region.
3. reg-pxl-ct: the number of pixels in a region = 9.
4. short-ln-dens-5: the results of a line extractoin algorithm that counts how many lines of length 5 (any orientation) with low contrast, less than or equal to 5, go through the region.
5. short-ln-dens-2: same as short-line-density-5 but counts lines of high contrast, greater than 5.
6. vedge-mn: measure the contrast of horizontally adjacent pixels in the region. There are 6, the mean and standard deviation are given. This attribute is used as a vertical edge detector.
7. vegde-sd: (see 6)
8. hedge-mn: measures the contrast of vertically adjacent pixels. Used for horizontal line detection.
9. hedge-sd: (see 8).
10. int-mn: the average over the region of (R + G + B)/3
11. rawred-mn: the average over the region of the R value.
12. rawblue-mn: the average over the region of the B value.
13. rawgrn-mn: the average over the region of the G value.
14. exred-mn: measure the excess red: (2R - (G + B))
15. exblue-mnn: measure the excess blue: (2B - (G + R))
16. exgrn-mean: measure the excess green: (2G - (R + B))
17. value-mn: 3-d nonlinear transformation of RGB. (Algorithm can be found in Foley and VanDam, Fundamentals of Interactive Computer Graphics)
18. sat-mn: (see 17)
19. hue-mean: (see 17)

The data is balanced, with 330 data points for each class. And there are no missing data.

1.2 Project overview.

The project is organized in the following way: In the section 2 a standard exploratory analysis is performed, with the goal of finding the structure of the data, patterns and problems that need to be addressed with preprocessing. In the section of data wrangling section 3.2 the data is reloaded and split into test/train data. The data is wrangled and then transformed with three different procedures, each transformation yielding a different dataset. Each of these datasets is analyzed with the same three algorithms and the results compared.

Section 3.1 explains briefly each algorithm and transformation used In each of the sections 3.3, 3.4 and 3.5 the three models are implemented, using in each section data with a different transformation. In 4 the results are presented discussed.

2 Data description and exploratory analysis.

As it was introduced in the previous section, the data consists of 2310 data points and 20 variables, of which one is a categorical response and the rest are quantitative variables. There are no missing values.

```
dim(segment)
```

```
## [1] 2310   20
```

```
any(is.na(segment))
```

```
## [1] FALSE
```

```
names(segment)
```

```
##  [1] "reg_cent_col"  "reg_cent_row"  "reg_pxl_ct"    "shrt_ln_dns_5"
##  [5] "shrt_ln_dns_2" "vedge_mn"      "vegde_sd"      "hedge_mn"
##  [9] "hedge_sd"      "int_mn"        "rawred_mn"     "rawblue_mn"
## [13] "rawgreen_mn"   "exred_mn"      "exblue_mn"     "exgreen_mn"
## [17] "value_mn"      "sat_mn"        "hue_mean"      "segment"
```

First let's see the summary statistics of the variables, as presented in table 1

Table 1: Summary statistics of predictors

	reg_cent_col	reg_cent_row	reg_pxl_ct	shrt_ln_dns_5	shrt_ln_dns_2	vedge_mn
mean	124.91	123.42	9	0.01	0.00	1.89
sd	72.96	57.48	0	0.04	0.02	2.70
median	121.00	122.00	9	0.00	0.00	1.22
minimum	1.00	11.00	9	0.00	0.00	0.00
maximum	254.00	251.00	9	0.33	0.22	29.22

	vegde_sd	hedge_mn	hedge_sd	int_mn	rawred_mn	rawblue_mn
mean	5.71	2.42	8.24	37.05	32.82	44.19
sd	44.85	3.61	58.81	38.18	35.04	43.53
median	0.83	1.44	0.96	21.59	19.56	27.67
minimum	0.00	0.00	0.00	0.00	0.00	0.00
maximum	991.72	44.72	1386.33	143.44	137.11	150.89

	rawgreen_mn	exred_mn	exblue_mn	exgreen_mn	value_mn	sat_mn	hue_mean
mean	34.15	-12.69	21.41	-8.72	45.14	0.43	-1.36
sd	36.36	11.58	19.57	11.55	42.92	0.23	1.55
median	20.33	-10.89	19.67	-10.89	28.67	0.37	-2.05
minimum	0.00	-49.67	-12.44	-33.89	0.00	0.00	-3.04
maximum	142.56	9.89	82.00	24.67	150.89	1.00	2.91

A first inspection of the results, shows that the variable *reg-pxl-ct* only has one value. This variable represents the number of pixels per instance and is, by construction a constant, so it will be discarded. Further analysis shows that variables *shrt-ln-dens-5* and *srht-ln-dens-2*, have all the data concentrated and preprocessing shows them as near zero variance. These variables will be removed from the data set in the wrangle section, before implementing the models.

```
nzv(segment)
```

```
## [1] 3 5
```

```
table(segment$reg_pxl_ct)
```

```
##
##    9
## 2310
```

```
table(segment$shrt_ln_dns_5)
```

```
##
##           0 0.11111111 0.22222222 0.33333334
##        2032         259          18           1
```

To explore the distributions of the predictors, the histograms of each one is plotted in figure 1.

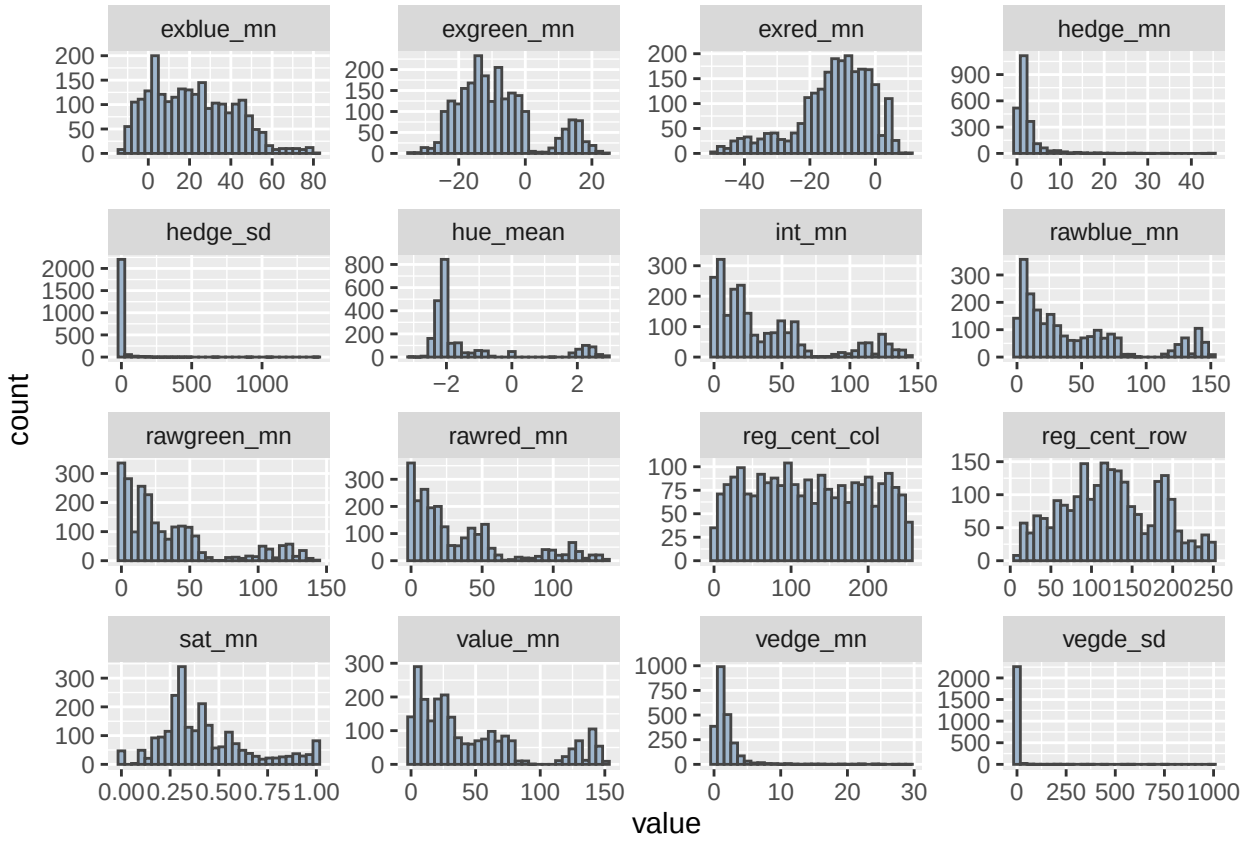


Figure 1: Histograms of predictors

The histograms show that variables *dedge_sd*, and *vedge_sd* are highly skewed. And some of the distributions are multimodal, which could indicate that the distribution of some variables are different for each class,

making it easier to separate them. This is further explored in the ridge plot, which shows the densities of each variable separated by class, in figure 2.

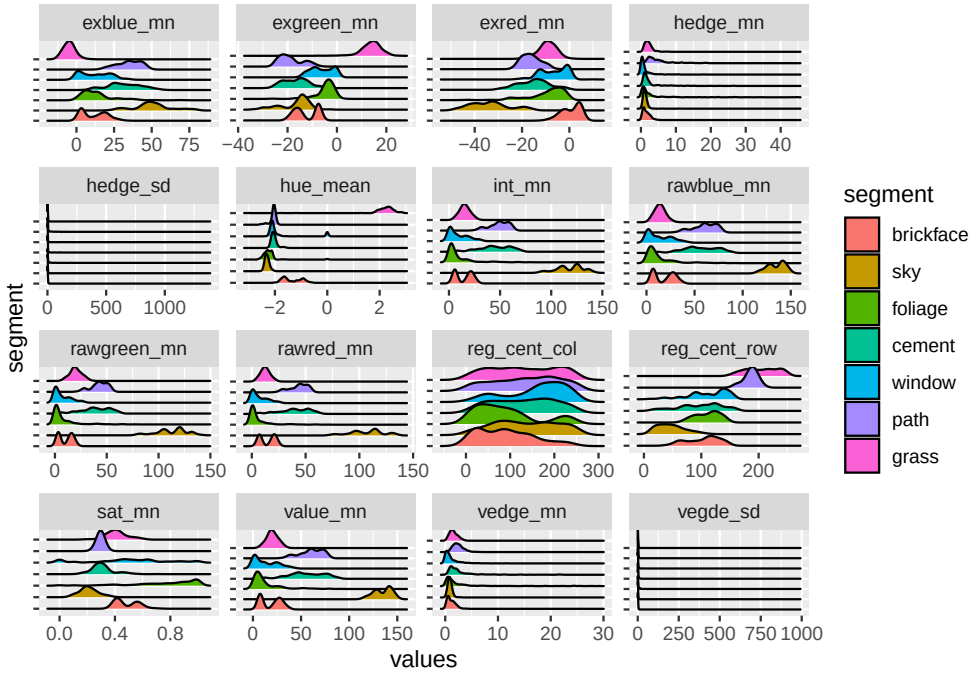


Figure 2: Densities of predictors by class

The first thing to notice in this plot is that the densities are shaped differently for different segments. For *grass* the density is bells shaped and and for most variables, centered away from the rest, so this class is expected to be the easiest to classify. The class *sky* is prefectly separated from the rest in the variable *rawgreen_mn* and is also expected to be easily classified. The density for *brickface* appears bimodal for most of the variables. The ridge plots for the variables *int_mn*, *rawred-mn*, *rawblue_mn* and *rawgreen_mean* appear almost identical, we expect these variables to be very correlated. This is further corroborated in the correlation plot in figure 3. Five variables have a correlation of almost 1 and there are others very highly correlated. Before attempting any model, these variables have to be dealt with, usually by removing them.

In many real life cases, the datasets may have dozens of variables, making it difficult to assess by looking at plots. Some preprocessing can be done to automatically deal with these variables. For example, in this case, we could use the commands from *caret* used for finding near zero variance variables, highly correlated and linear combinations.

```
#correlation matrix
corel <-segment %>% select(-segment) %>%
    cor()
#find programatically correlations and problematic variables
nzv(segment)
```

```
## integer(0)
```

```
findCorrelation(corel)
```

```
## [1]  9 14  7  8
```

```
findLinearCombos(corel)
```

```
## $linearCombos
```

```
## $linearCombos[[1]]
## [1] 10  7  8  9
##
## $linearCombos[[2]]
## [1] 11  7  8
##
## $linearCombos[[3]]
## [1] 12  7  9
##
## $linearCombos[[4]]
## [1] 13  7  8  9
##
##
## $remove
## [1] 10 11 12 13
```

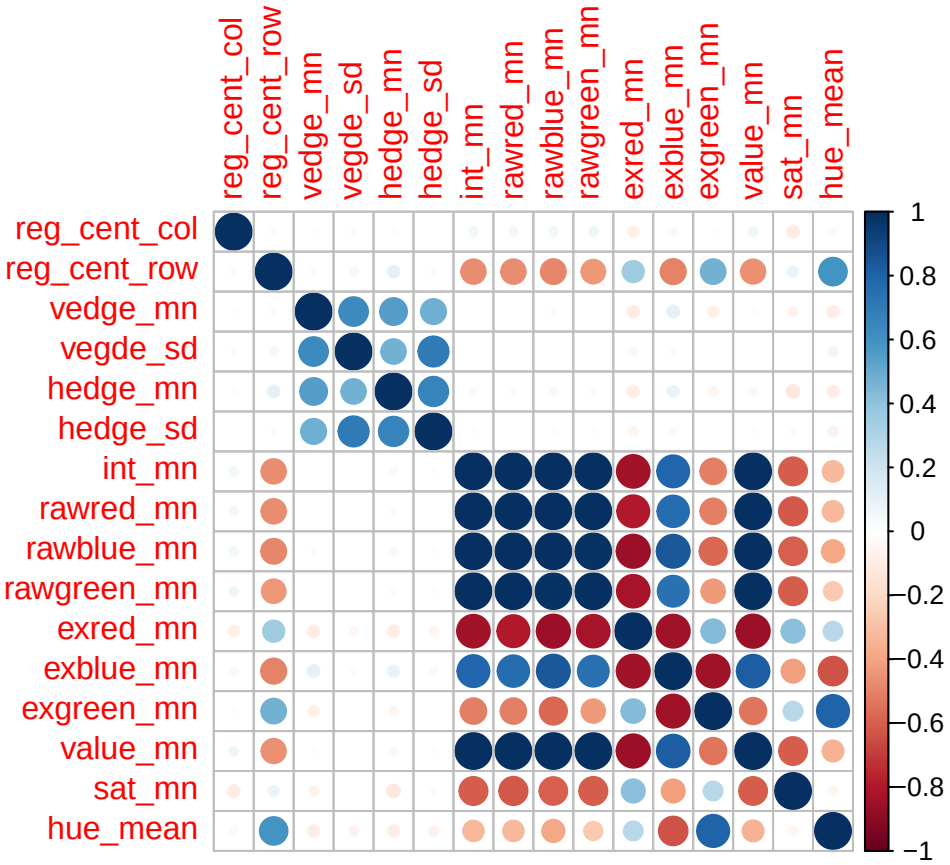


Figure 3: Correlation plot of predictors

3 Analysis and methods used.

In this section we seek the best procedure for classification of the *segment* variable. The algorithms used are knn (nearest neighbors), classification trees and random forest. The data is split into a train/test set with the 20% observations into the training set. The training set is subjected to three different preprocessing

procedures: removal of highly correlated variables, removal of those variables and center/scale, and finally preprocess by principal components with all the variables included. The three training sets obtained are trained in the three algorithms above mentioned. The algorithms are compared through the accuracy in the test sets and the results compared.

3.1 Brief explanation of the models and transformations.

The algorithms used for segmentation are knn, classification trees and random forest. A brief explanation of each is presented below.

The knn algorithm is very easy to understand and very intuitive. To find an estimate in for a point x in the test set:

- Select a number of neighbors k
- For a given point x in the test set, find the k points in the TRAINING set that are closer in eculidean distance
- Find the responses for those k points
- Assign to x the response with the largest proportion.

To find the optimal k, reslampling methods such as cross validation are used in the training set. A very small k results in a more flexible boundary, but more prone to high variance.

Tree based models.

Both regression and classification tree methods are based ona successive partition of the predictor space into N non-overlapping regions $R_1, R_2 ... R_N$. At every split, all the predictors are considered for the split. Each split is made at the predictor and value that results in the largest reduction of the RSS for regression and Gini index for classification.

The process is repeated N times in order to get N splits, where N is given by a stopping rule. The stopping rule, called Cp, complexity parameter is given by a tradeoff on how big the reduction of the metric is, vs how complex the tree gets. The predictions are then obtain as either the mean value of the prediction or the most common class of the predicion in each partition, depending whether it is a regression or classification problem.

Random forest are based on tree based models, as the result of an average of several complex trees, each built on bootstrapped sample of the data instead of the data itself. The trees tend to be very correlatad. Random forest corrects this by allowing each split to consider only a subset of m from the p predictors. This number m is a tuning parameter in the algorithm and is found with cross-validation or bootstrap (called *mtry* in caret).

This preprocessing process includes, among many other steps, cleaning the data, dealing with missing values and in many cases transforming the variables. It also involves dealing with potential problems diagnosed in the exploratory analysis, such as highly correlated variables or linear combinations.

Transforming the variable consists on applying a function to the predictors that result in a new variable to be used for the analisis, such as centering and scaling, transformations to reduce skewness and the use of linear combinations of predictors, such as principal components analysis. The transformations needed to get the best performance depend in part on each model. For example, some models are based on the assumption of normally distributed data, and those can benefit from transformations that normalize the data. Some algorithms require the covariance matrix to be invertible, so they will simply not run unless there are no high correlations or linear dependencies. Some transformations result in fewer predictors. This usually has the advantage of shorter computation time.

In this project three different transformations are performed in the data and each transformed data analyzed with the three algorithms. The exploratory analysis showed several highly correlated variables, so the first transformation performed is removing the correlated variables (keeping one of them). The second transformation is, additionally, centering and scaling the data. This preprocessing is usually recommended for the knn algorithm. The third preprocessing used is principal component analysis, using all the variables.

This transformation includes the center-scale and takes care of the correlations and linear dependencies of the data.

3.2 Data wrangling and preprocessing.

First thing to be done is to reload the data, as it was altered for the exploratory analysis. The data doesn't have any missing data. In the exploration, three variables that are almost non-zero variance were detected. First these are eliminated.

```
load("data/segment.RData")
segment <- segment %>% select(-reg_pxl_ct, -shrt_ln_dns_5, -shrt_ln_dns_2)
```

The data is now partitioned into a test-train set. This is done before the preprocessing because the test data has to be preprocessed according to the parameters determined in the train set. For example, centering and scaling in the test set should be done according to the mean and variance found in the training set.

```
####### train/test split

set.seed(100)
trainIndex <- createDataPartition(segment$segment,
                                  p=0.8,
                                  times=1,
                                  list=F)
segment_train <- segment[trainIndex,]
segment_test <- segment[trainIndex,]
```

The next step is to create three versions of our train/test data, each one the result of a different preprocessing:

3.2.1 Remove highly correlated variables.

The first preprocess is a simple removal of the three of the four variables that are almost perfectly correlated. This dataset will be labeled train_inc and test_inc.

```
train_inc <- segment_train %>% select(-rawred_mn, -rawblue_mn, -rawgreen_mn, -value_mn)
test_inc <- segment_test %>% select(-rawred_mn, -rawblue_mn, -rawgreen_mn, -value_mn)
```

3.2.2 Center and scale.

Here we take the data without the highly correlated variables and center-scale it. The data are labeled train_cs and test_cs.

```
# Center and scale
cent_scale <- preProcess(test_inc, method = c("center", "scale"))
train_cs <- predict(cent_scale, train_inc)
test_cs <- predict(cent_scale, test_inc)
```

3.2.3 Preprocess with principal components.

Before performing the actual transformation, first let's look at the results of a pca analysis. In the scree plot (fig 4), we can see that almost half of the variance is explained only by the first principal component.

We finally apply the principal component preprocessing, choosing the number of components that explian 95% of the variance. Only eight predictors out of the 16 are needed in order to explain 95% of the variance.

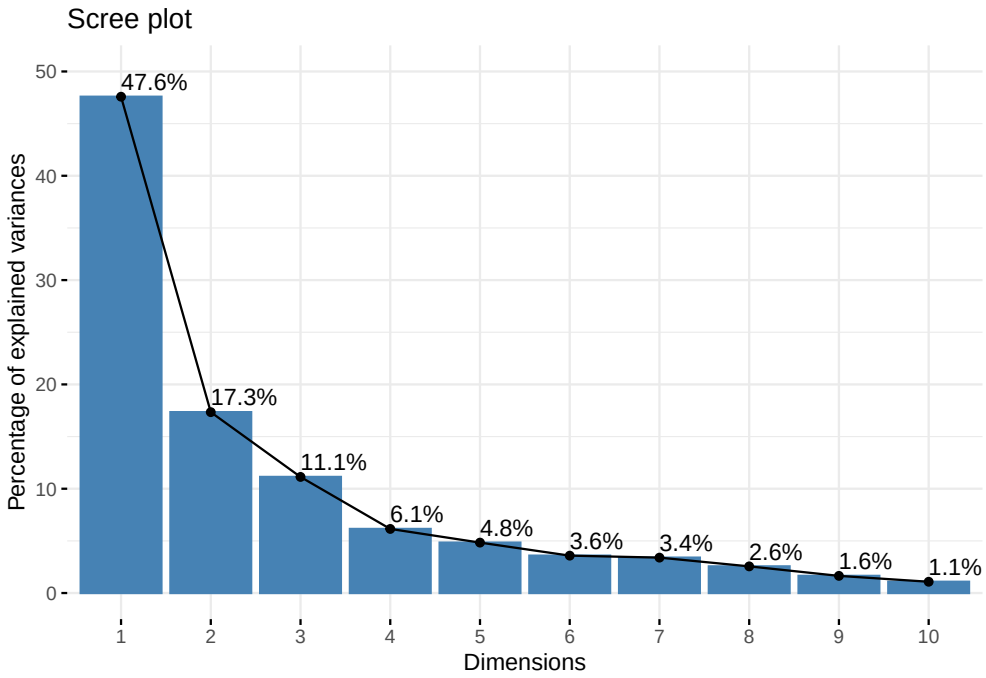


Figure 4: Scree plot

```
# Preprocess data with caret
prep <-preProcess(segment_train, method="pca", thresh=0.95)
prep
```

```
## Created from 1848 samples and 17 variables
##
## Pre-processing:
##   - centered (16)
##   - ignored (1)
##   - principal component signal extraction (16)
##   - scaled (16)
##
## PCA needed 8 components to capture 95 percent of the variance
```

```
# train/test sets with transformed variables
train_pca <-predict(prep,segment_train)
test_pca <-  predict(prep, segment_test)
```

3.3 Model building with first pre-process procedure (removal of highly correlated predictors).

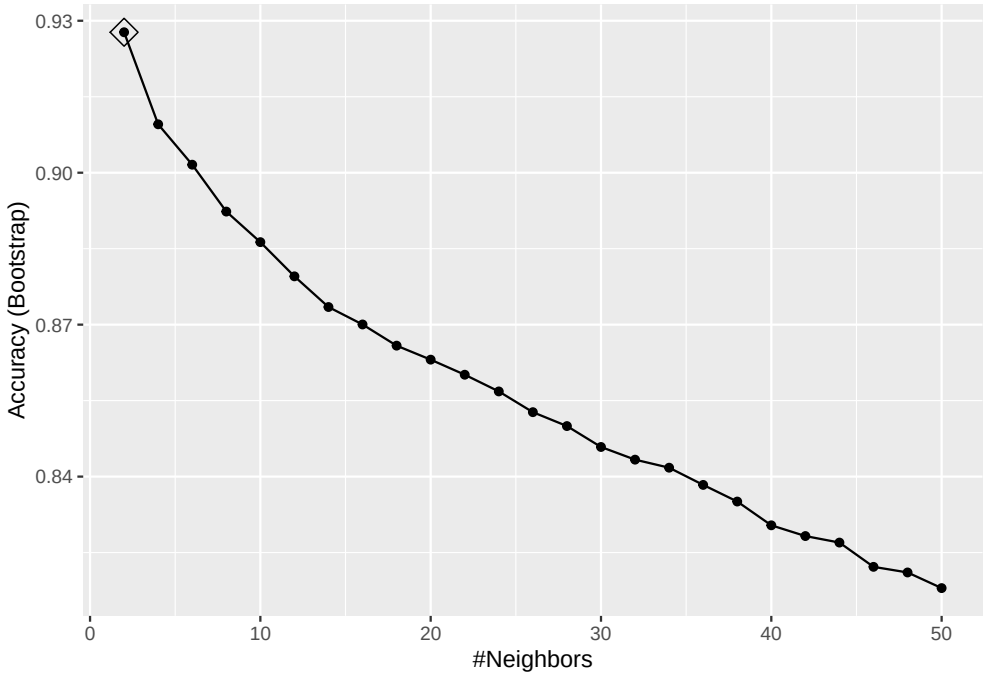
In this section, the three models introduced earlier are implemented in the data with the correlated variables removed, and the results are compared.

3.3.1 knn (k-nearest neighbors).

The first model trained is a k-nearest model. The model is trained with caret, evaluating with boostrap the optimal number of k (neighbors).

```
#train model on a grid of different values for k
set.seed(100)
train_knn <- train(segment ~ .,
                   method = "knn",
                   data = train_inc,
                   tuneGrid = data.frame(k = seq(2, 50, 2)))

#plot the results of bootsrap on each k
ggplot(train_knn, highlight = TRUE)
```



Now, let's evaluate the table in the test set and find the accuracy.

```
accu_knn1 <-confusionMatrix(predict(train_knn, test_inc, type = "raw"),
                            test_inc$segment)$overall["Accuracy"]

cf <-confusionMatrix(predict(train_knn, test_inc, type = "raw"),
                     test_inc$segment)$table

#number of missclassified instances.
misclas <-(1-accu_knn1[1])*nrow(test_inc)
```

Table 4: Confusion matrix of knn

	brickface	sky	foliage	cement	window	path	grass
brickface	262	0	1	2	2	0	0
sky	0	264	0	0	0	0	0
foliage	0	0	247	1	9	0	0
cement	0	0	0	253	4	0	0

	brickface	sky	foliage	cement	window	path	grass
window	2	0	15	8	248	0	0
path	0	0	1	0	1	264	0
grass	0	0	0	0	0	0	264

The plot of the model shows an optimal number of neighbors of 2. The accuracy of this first algorithm is `0.9772727`, there are still `42` cases missclassified, in the test set. The confusion table 4 shows what we were expecting, the *sky* and *grass* would be the easiest to predict, given that those classes had a distribution of values, at least in one predictor, well separated from the rest. The classes *foliage* and *window* are the ones that had more misses. To gain a bit of more insight of where the missclassifications are, a boxplot of two predictors is presented in figure 5, with the values of the missclassified instances labeled.

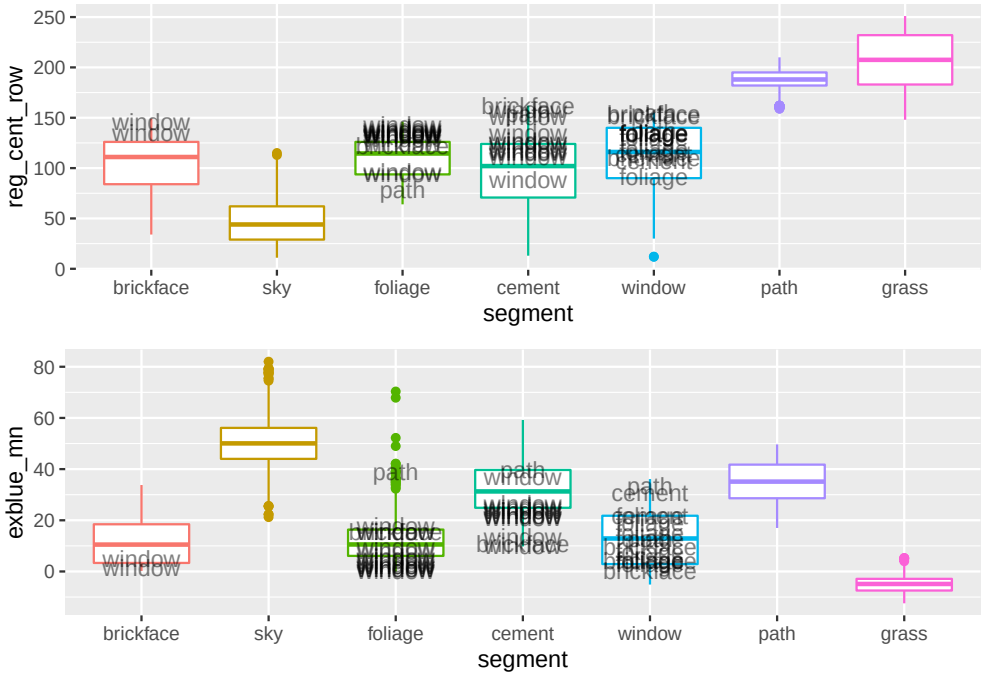


Figure 5: Missclassified by knn

The values of *reg_cent_row* for classes window and foliage are very similar and these 2 get very confused. Various values for cement were wrongly labeled as window. These instances had values for *exblue_mn* that were outside the IQR for cement but close to the median for window.

3.3.2 Classification tree.

Now a classification tree will be used to classify the data. The caret package is used and several values of the complexity parameter are tested with bootstrap.

```
set.seed(101)
train_rpart <- train(segment ~ .,
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                     data = train_inc)
accu_rpart1 <-confusionMatrix(predict(train_rpart, test_inc, type = "raw"),
                              test_inc$segment)$overall["Accuracy"]
```

One interesting thing about trees is that the results are very intuitive and some things can be learned about the data by studying the tree plots. In figure 6 we can see that the first split is done in the variable *int_mn* (average over the region), and it perfectly classifies the class *sky*. It clearly illustrates the order of importance of the predictors in the classification.

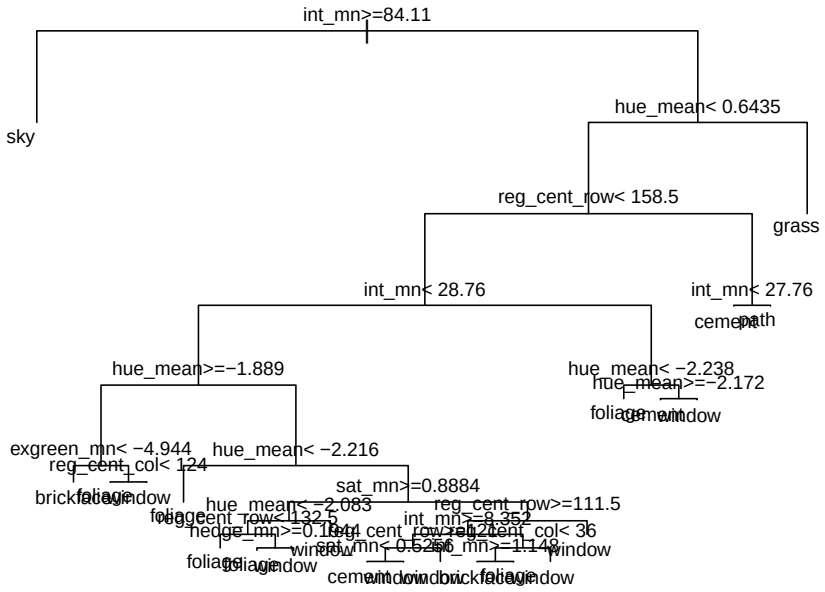


Figure 6: Classification tree for best tune.

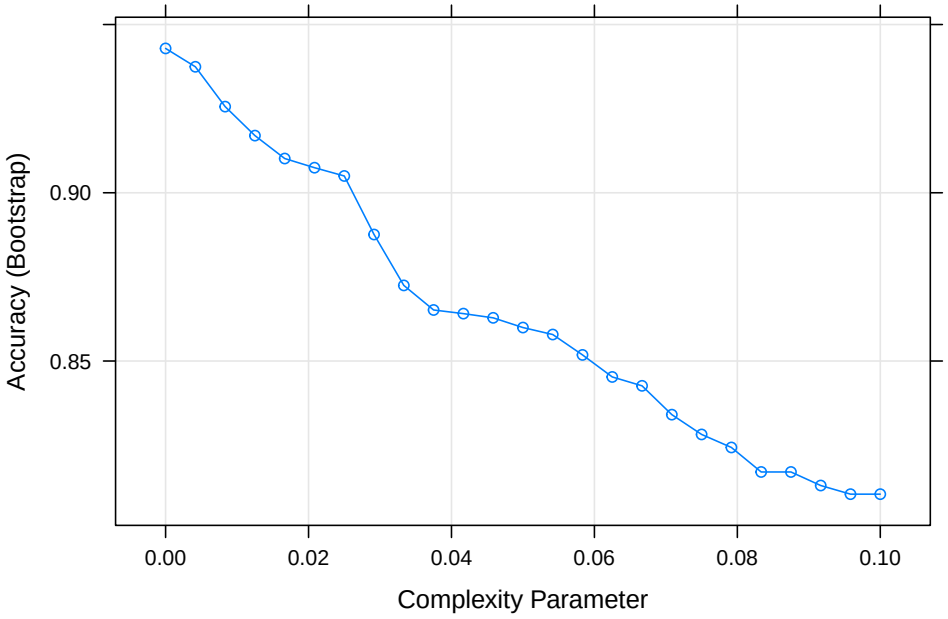


Figure 7: Bootstrap estimated error as a function of complexity parameter

The plot 7 shows the estimated errors (by cross-validation) as a function of each complexity parameter tested. The cp used in the final model is `0`, which means that the best tune is a full grown tree. The accuracy is `0.9713203` which is a bit worse than with knn. Even if the accuracy of the classification trees is actually a bit worse than the knn, it is a good model to help gain insights of patterns in the data.

3.3.3 Random forest.

For random forest, the tuning parameter is *mtry*, the number of variables from which the algorithm randomly is allowed to choose for next split. In our case we have 12 variables left, so the tuning parameters to be tested will be integers between 1 and 12. The algorithm is timed in order to compare it later on. The bootstrapped estimate of the accuracy is plotted against values of *mtry*, in figure 8.

```
set.seed(10)
st <- Sys.time()

grid <- data.frame(mtry = 1:12)
control <- trainControl(method="cv", number = 5)

train_rf <- train(segment~.,
                  method = "rf",
                  ntree = 250,
                  tuneGrid = grid,
                  trControl = control,
                  data = train_inc)
end <- Sys.time()
time_rf <-  end-st

predictions <- predict(train_rf, newdata=test_inc)

cftree <-confusionMatrix(predictions, test_inc$segment)
accu_rf1 <- cftree$overall["Accuracy"]
```

The best tune for the rendom forest is `3`. The accuracy for this algorithm is `0.9994589`, which is a big improvement. Although it is not as intuitive as classification trees, it is certainly more powerful. Even though we don't have a diagram to clearly indicate the order in which the splits are made and in which variables, we can still find the importance of each variable with a variable importance plot, indicating the gini coefficient reduction for each variable, in figure 9. The most important variables are *int_mn*, *hue_mean* and *reg_cent_row*, which is consistent with what we observed in the classification tree, as the first 3 splits were made in those variables.

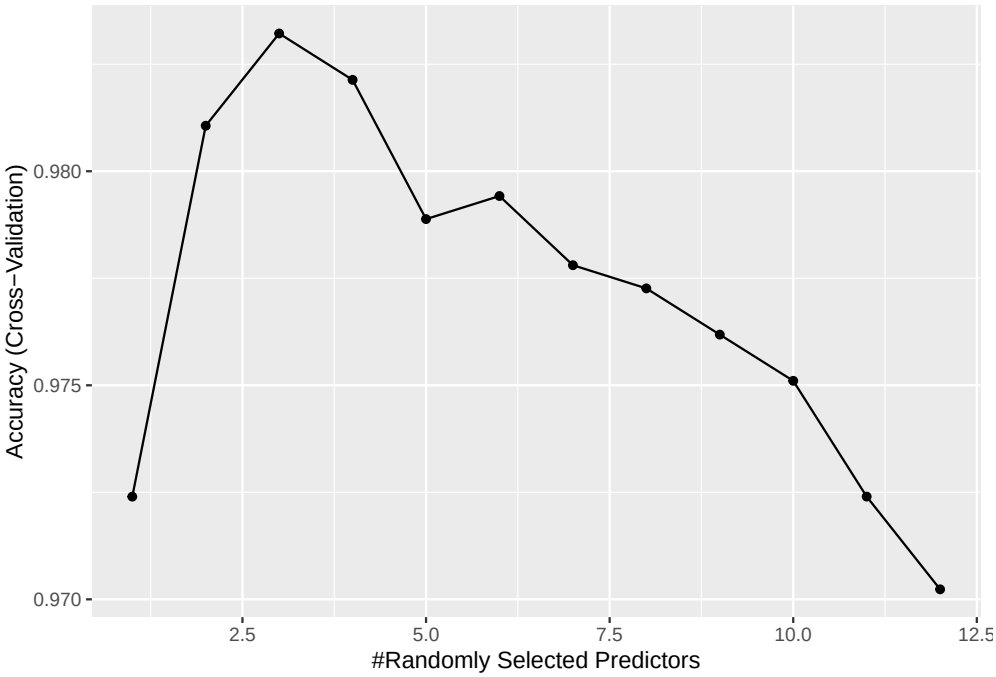


Figure 8: Cross-validation error as a function of tuning parameter

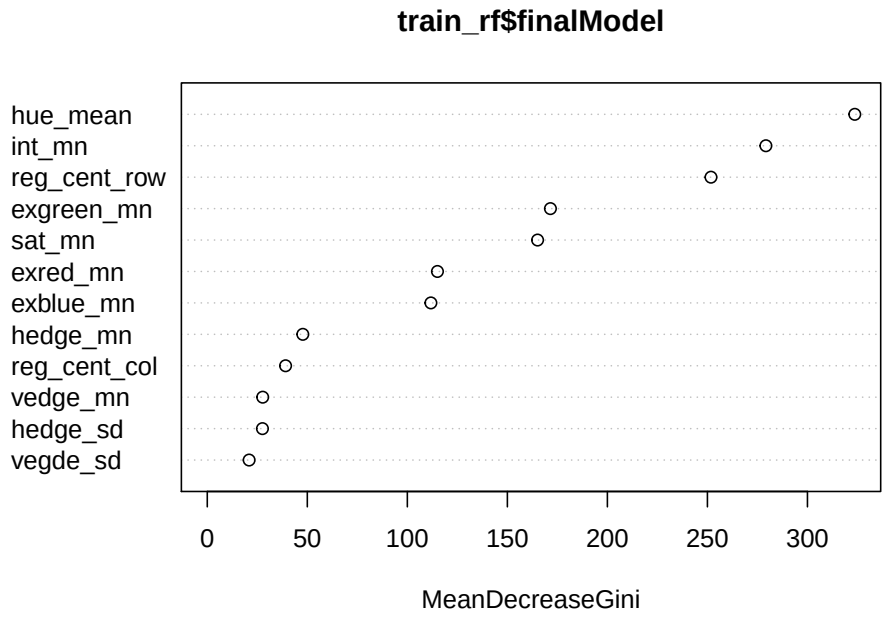


Figure 9: Variable importance in random forest

3.4 Model building with third pre-process procedure (centered and scaled).

In this section, the three models introduced earlier are implemented in the data with the correlated variables removed, and center scaled. As it is very similar to the previous section, only the results for the best tune and not the plots will be presented. (These can be found in the code file).

3.4.1 knn (nearest neighbor), with centered-scaled data.

We repeat the procedure as before, but with the centered and scaled data.

```
set.seed(100)
train_knn2 <- train(segment ~ .,
                    method = "knn",
                    data = train_cs,
                    tuneGrid = data.frame(k = seq(2, 50, 2)))
#accuracy
accu_knn2 <-confusionMatrix(predict(train_knn2, test_cs, type = "raw"),
                            test_cs$segment)$overall["Accuracy"]
```

The accuracy for the knn algorithm center-scaled data is `0.9816017`, an improvement over the accuracy with non centered-scale `0.9772727`. This result is consistent with the knowledge that k-nearest neighbors perform better with centerd and scaled data.

3.4.2 Classification tree, with center-scaled data.

Now the classification tree algorithm is applied to the centered and scaled data:

```
###### Classification tree model on center-scale

set.seed(101)
train_rpart2 <- train(segment ~ .,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                      data = train_cs)


accu_rpart2 <-confusionMatrix(predict(train_rpart2, test_cs$test, type = "raw"),
                              test_cs$segment)$overall["Accuracy"]
```

The accuracy for the classification tree with center-scaled data is `0.9713203`, exactly the same result as with non centered-scaled `0.9713203`.

3.4.3 Random forest , with center-scaled data.

The random forest algorithm is applied to the center-scaled data.

```
set.seed(10)
st <- Sys.time()

grid <- data.frame(mtry = 1:12)
control <- trainControl(method="cv", number = 5)
train_rf2 <- train(segment~.,
                   method = "rf",
                   ntree = 250,
```

```
                  tuneGrid = grid,
                  trControl = control,
                  data = train_cs)

end <- Sys.time()
time_rf2 <-  end-st

predictions <- predict(train_rf2, newdata=test_cs)

cftree2 <-confusionMatrix(predictions, test_cs$segment)
accu_rf2 <- cftree$overall["Accuracy"]
```

As expected, the results for this algorithm is the same as the previous one, with an accuracy of `0.9994589`

3.5 Model building with third pre-process procedure (principal compontents).

In this section, the three models introduced earlier are implemented in the data preprocessed with principal components. (Principal components applied to all the variables, including the highly correlated). Eight principal components are used to preserve 95% of the variance, so the algorithms in this section are being fit in four fewer variables than in the previous two tranformations. The expected advantage is less computation time.

3.5.1 K- nearest neighbors with PCA preprocessed data.

The pca preprocessed data is analyzed with knn algorithm:

```
set.seed(100)
train_knn3 <- train(segment ~ .,
                    method = "knn",
                    data = train_pca,
                    tuneGrid = data.frame(k = seq(2, 50, 2)))

#accuracy
accu_knn3 <-confusionMatrix(predict(train_knn3, test_pca, type = "raw"),
                            test_pca$segment)$overall["Accuracy"]
```

the accuracy is `0.9810606`, which is the same as the one for center-scale. Using PCA may have the advantage of using a smaller number of predictors, therefore making the computation faster.

3.5.2 Classification tree with PCA preprocessed data.

The pca preprocessed data is analyzed with classification tree algorithm:

```
set.seed(101)

train_rpart3 <- train(segment ~ .,
                      method = "rpart",
                      tuneGrid = data.frame(cp = seq(0.0, 0.1, len = 25)),
                      data = train_pca)

accu_rpart3 <-confusionMatrix(predict(train_rpart3, test_pca$test, type = "raw"),
                              test_pca$segment)$overall["Accuracy"]
accu_rpart3
```

```
##  Accuracy
## 0.9426407
```

For this algorithm, the transformation resulted in an accuracy of `0.9426407`, which is lower than the previous results of `0.9713203`.

3.5.3 Random forests with PCA pre processed data.

The last algorithm implemented with the PCA processed data is a random forest. Like it was done in the previous implementations of this algorithm, the process is timed. In this case, the maximum value for *mtry* is 8 instead of 12, because we only have 8 PCA components.

```
set.seed(10)
st <- Sys.time()

grid <- data.frame(mtry = 1:8)
control <- trainControl(method="cv", number = 5)
train_rf3 <- train(segment~.,
                   method = "rf",
                   ntree = 250,
                   tuneGrid = grid,
                   trControl = control,
                   data = train_pca)

end <- Sys.time()
time_rf3 <-  end-st

predictions <- predict(train_rf3, newdata=test_pca)
cftree3 <-confusionMatrix(predictions, test_pca$segment)
accu_rf3 <- cftree$overall["Accuracy"]
```

For this algorithm, the transformation resulted in an accuracy of `0.9994589`, which is the same as with previous transformations.

4 Results and discussion.

In the previous sections the data set was split into a train and a test set, and three different preprocessing procedures applied. To each of these 3 preprocessed data, the same three algorithms were implemented to classify the variable *segment*. Additionally, the random forest algorithm was timed. The results are presented in table 5.

Table 5: Summary or results

	knn	tree	Random_forest	RF_time
Highly correlated removed	0.9772727	0.9713203	0.9994589	23.38020 secs
Center-scale	0.9816017	0.9713203	0.9994589	22.82422 secs
PCA	0.9810606	0.9426407	0.9994589	14.05636 secs

The results indicate that:

- The random forest algorithm outperforms the knn and classification tree. It only missclassified `1`.
- Center and scaling improved the results for *knn* but not either of the tree-based.

- Data with principal component analysis resulted in the same performance for knn and random forests. For classification tree resulted in a much worse accuracy.
- The random forest algorithm resulted in the same accuracy with PCA, and the computation was about 30% faster.

Random forests results in the best performance overall. It generally has the disadvantage of being slower than the other two. However, using a dimension reduction preprocessing such as PCA results in much faster computation times. Principal components have the disadvantage of not presenting the results related to the data directly, which may pose a problem when trying to gain insights from the data, but is less of a problem when predicting a response. Classification trees offer very intuitive results which help understand the different relationships with the data, but the performance is poorer. Variable importance in the random forest algorithm helps understand the data as well.

5 Conclusions.

The objective of the project was to find the algorithm that best classifies the 7 classes in the image dataset and to emphasize the importance of preprocessing, especially dealing with highly correlated data and appropriate transformation. For that effect, three different transformations were applied to the data. We can conclude that the best procedure for this dataset was random forests with PCA transformed predictors. This combination of algorithm and transformations yields the highest accuracy while having the shortest computation time. Although other algorithms and transformations offer more possibility of insight of in the behavior and patterns of the data, as the objective of this project is to find the best classification possible.

It can also be concluded that pre processing and transformations an extremely important step in predictive analysis. It is important, however, to study what kind of transformation is best suited for each algorithm. In the project, knn performed better with center-scaling. Both tree based methods (classification tree and random forest) were unaffected by this transformation. PCA was the best for random forest since it cut the computaton time.