Assignment 3: Machine Learning

Matteo Melis (40324932)

Introduction

In this report, we will utilise the features that we created in Assignment 2 to address classification issues through machine learning. The primary objective will be to create and assess classifiers for the image data to determine if the models can accurately predict the class labels for new images.

Section 1

Section 1 requires us to use the dataset acquired from Assignment 2 to create prediction models that classify an image as living or non-living. We will report on the accuracy and other key aspects of these models.

Section 1.1

To begin, the 112 items will be partitioned into a training set and test set. A total of 24 items will be randomly selected to form a test set via stratifed sampling. The four most significant features, determined from the previous assignment, will be chosen and used to train a logistic regression model. The model will then be utilised to predict whether the images in the test data belong to the "living" or "non-living" category.

```
library(rstudioapi)
library(stringr)
library(splitstackshape)
library(caret)
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
csv_df <- read.csv(file = "40324932_features.csv")

csv_df$livingThing <- "no"
csv_df$livingThing[csv_df$label %in% c("banana", "cherry", "lemon",
    "tree")] <- "yes"
csv_df$livingThing <- as.factor(csv_df$livingThing)

csv_df$key <- paste(csv_df$label, csv_df$indexs, sep = "")  #create a unique identifier
test_data <- stratified(csv_df, "labels", 3)  #stratified sample of 3 of each image
training_data <- csv_df[!(csv_df$key %in% test_data$key), ]  #take everything not in test data

# remove the key column again
csv_df <- subset(csv_df, select = -c(key))
test_data <- subset(test_data, select = -c(key))
training_data <- subset(training_data, select = -c(key))

# fit a logistic regression model using 4 significant
# features
glmfit <- glm(livingThing ~ rows_with_1 + custom + cols_with_1 +
    eyes, data = training_data, family = "binomial")
summary(glmfit)
```

```
##
## Call:
## glm(formula = livingThing ~ rows_with_1 + custom + cols_with_1 +
##     eyes, family = "binomial", data = training_data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.39160  -0.22005   0.00047   0.09864   3.14103
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  13.7783     6.9239   1.990  0.04660 *
## rows_with_1  -1.1326     0.4205  -2.693  0.00707 **
## custom        0.2038     2.7194   0.075  0.94027
## cols_with_1  -0.3378     0.3871  -0.873  0.38281
## eyes         -8.5926     3.3667  -2.552  0.01070 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 121.994  on 87  degrees of freedom
## Residual deviance:  18.849  on 83  degrees of freedom
## AIC: 28.849
##
## Number of Fisher Scoring iterations: 8
```

For the logistic regression model, the four features with the lowest p-values on the difference of mean values between living and non-living things from the previous assignment were selected. These features are rows_with_1, cols_with_1, eyes, and curvature/custom. The model was fitted using the training data and we will generate a confusion matrix to display the predicted class versus actual class for each of the 24 items in the test set.

```
library(pander)
# test the prediction model on the test_data
test_data[["predicted_val"]] = predict(glmfit, test_data, type = "response")
test_data[["predicted_class"]] = "no"
test_data[["predicted_class"]][test_data[["predicted_val"]] >
    0.5] = "yes"

# check which items were correct
correct_items = test_data[["predicted_class"]] == test_data[["livingThing"]]
correct_items
```

```
##  [1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
# create a confusion matrix
cm = confusionMatrix(table(test_data[["predicted_class"]], test_data[["livingThing"]]))

names(dimnames(cm$table)) <- c("Predicted", "Actual")
pander(ftable(cm$table), caption = "Confusion Matrix of our model on the test dataset",
    style = "rmarkdown")
```

Table 1: Confusion Matrix of our model on the test dataset

	Actual	no	yes
Predicted			
no		12	1
yes		0	11

Based on the results, the logistic regression model accurately predicted 23 out of the 24 images in the test data when a decision threshold of 0.5 was used. The printout of correct_items tells us that the second image in the test set was the only incorrect prediction.

```
cat(test_data$labels[2], test_data$index[2])
```

```
## banana 5
```

The model predicted banana_05 to be a non-living thing.

```
kable(cbind(csv_df[5, c("labels", "rows_with_1", "cols_with_1",
    "eyes", "custom")]), caption = "Banana 05 features")
```

Table 2: Banana 05 features

	labels	rows_with_1	cols_with_1	eyes	custom
5	banana	0	1	2	1.034

The misclassification of banana_05 can be attributed to some unexpected features of that particular drawing, since the banana has two eyes instead of the expected one. This outlier in the data may have led the model to predict it as a non-living object. It's worth noting that a models performance can be evaluated based on various factors, not just its accuracy. Let's delve deeper into the characteristics of this model below.

```
methods <- c()
# Caclulate TP, FP, TN, FN
TP <- cm$table[2, 2]
FP <- cm$table[2, 1]
TN <- cm$table[1, 1]
FN <- cm$table[1, 2]
# calculate the difference performance metrics and store in
# a list
accuracy <- sum(correct_items)/nrow(test_data)
methods <- append(methods, accuracy)

true_positive <- TP/(TP + FN)
methods <- append(methods, true_positive)

false_positive <- FP/(FP + TN)
methods <- append(methods, false_positive)

precision <- TP/(TP + FP)
methods <- append(methods, precision)

F1 <- 2 * ((precision * true_positive)/(precision + true_positive))
methods <- append(methods, F1)

# combine results in a dataframe
```

```
results <- data.frame(c("Accuracy", "True Postive Rate/Recall",
    "False Positive Rate", "Precision", "F1 Score"), methods)

kable(results, col.names = c("Evaluation Method", "Value"), caption = "A table showing the value of dif
```

Table 3: A table showing the value of different methods of evaluating our model:

Evaluation Method	Value
Accuracy	0.9583333
True Postive Rate/Recall	0.9166667
False Positive Rate	0.0000000
Precision	1.0000000
F1 Score	0.9565217

Note: In the following evaluations, T = True, F = False, P = Positive and N = Negative. For example FN denotes a False Negative.

The model's accuracy is approximately 96%, indicating that the chosen predictor variables are effective since the model made only one incorrect classification on the test data.

The term True Positive Rate, also called Recall or Sensitivity, is defined as the probability that an actual positive will test positive. It is calculated as follows: $\frac{\text{TP}}{\text{TP + FN}} = \frac{11}{11+1} = 0.917$. The models only mistake was a false negative when it predicted a banana to be a non-living thing so this value makes sense given the predictions made by the model.

The false positive rate is calculated as the ratio between the number false positives and the total number of actual negative events. It is calculated as follows: $\frac{\text{FP}}{\text{FP + TN}} = \frac{0}{0+12} = 0$. Since our model never predicted a non-living thing to be a living thing our false positive rate will be 0. This also tells us our specificity would be 1.

Precision is described as: $\frac{\text{TP}}{\text{TP + FP}} = \frac{12}{12+0} = 1$. This tells us our model was perfectly precise, meaning when it predicted a living thing, it was always correct.

F1 Score is described as: $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision + recall}} = 0.96$. A high F1 score tells us that both our precision and recall are high, this along with accuracy are useful evaluation metrics for this model since the number of false negatives and false positives made by the model are equally important to us.

Section 1.2

In this section we will use the same 4 features as our previous model but with a different approach. We will use 7-fold cross validation over all 112 items and note the differences between the two models.

```
# logistic regression model using 7 fold cross validation
kfoldsk = 7
train_control <- trainControl(method = "cv", number = kfoldsk,
    savePredictions = TRUE, classProbs = TRUE)

# train the model on the entire dataset
model <- train(livingThing ~ rows_with_1 + custom + cols_with_1 +
    eyes, data = csv_df, trControl = train_control, method = "glm",
    family = "binomial")

# summarize results
print(model)
```

```
## Generalized Linear Model
##
## 112 samples
##   4 predictor
##   2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (7 fold)
## Summary of sample sizes: 96, 96, 96, 96, 96, 96, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9464286  0.8928571
```

```
# create confusion matrix
cm = confusionMatrix(table(model$pred$yes >= 0.5, model$pred$obs ==
    "yes"))

names(dimnames(cm$table)) <- c("Predicted", "Actual")
pander(ftable(cm$table), caption = "Confusion Matrix of our 7-fold model on 112 image dataset",
    style = "rmarkdown")
```

Table 4: Confusion Matrix of our 7-fold model on 112 image dataset

	Actual	FALSE	TRUE
Predicted			
FALSE		54	4
TRUE		2	52

The confusion matrix generated by the model shows that there were 4 false negatives and 2 false positives, while the remaining 106 predictions were correct. Let us investigate the 6 incorrect predictions further:

```
# extract the incorrect predictions and store their details
# in a df
incorrect_preds <- model$pred[model$pred$pred != model$pred$obs,
    ]
incorrect_df <- cbind(csv_df[incorrect_preds$rowIndex, c("labels",
    "rows_with_1", "custom", "cols_with_1", "eyes")], incorrect_preds[,
    c("pred", "obs", "yes", "no")])

kable(incorrect_df, caption = "The incorrect predictions made by our model")
```

Table 5: The incorrect predictions made by our model

	labels	rows_with_1	custom	cols_with_1	eyes	pred	obs	yes	no
20	cherry	5	0.697	0	1	no	yes	0.4213849	0.5786151
5	banana	0	1.034	1	2	no	yes	0.1218467	0.8781533
55	golfclub	4	0.851	1	1	yes	no	0.8102805	0.1897195
17	cherry	0	0.511	10	1	no	yes	0.3446926	0.6553074
25	cherry	9	0.404	0	1	no	yes	0.0000009	0.9999991
54	golfclub	5	0.905	0	1	yes	no	0.7480834	0.2519166

It seems our model misclassified some cherries and golfclubs. It also made an error on the outlier banana we saw earlier however we can ignore this. Upon examination of the features file, we can deduce the following:

```
means <- c()
# find the means of all key features of cherries and
# golfclubs
means <- append(means, sum(csv_df[15:28, ]$rows_with_1)/14)
means <- append(means, sum(csv_df[43:56, ]$rows_with_1)/14)
means <- append(means, sum(csv_df[15:28, ]$cols_with_1)/14)
means <- append(means, sum(csv_df[43:56, ]$cols_with_1)/14)
means <- append(means, sum(csv_df[15:28, ]$eyes)/14)
means <- append(means, sum(csv_df[43:56, ]$eyes)/14)
means <- append(means, sum(csv_df[15:28, ]$custom)/14)
means <- append(means, sum(csv_df[43:56, ]$custom)/14)

# create df of the means
results <- data.frame(c("Cherries rows_with_1", "Golfclubs rows_with_1",
    "Cherries cols_with_1", "Golfclubs cols_with_1", "Cherries eyes",
    "Golfclub eyes", "Cherries custom", "Golfclub custom"), means)

kable(results, col.names = c("Feature", "Mean Value"), caption = "A table describing
      the mean value of each predictor variable for golfclubs and cherries:")
```

Table 6: A table describing the mean value of each predictor variable for golfclubs and cherries:

Feature	Mean Value
Cherries rows_with_1	1.6428571
Golfclubs rows_with_1	10.4285714
Cherries cols_with_1	1.2857143
Golfclubs cols_with_1	0.7857143
Cherries eyes	1.0000000
Golfclub eyes	1.0000000
Cherries custom	0.6807143
Golfclub custom	0.6990000

The analysis above reveals that the features cols_with_1, eyes, and custom have similar values for both cherries and golf clubs, making rows_with_1 the only distinguishing feature for classification. Investigating the cases where it made a wrong prediction we can see that in each case it is due to the rows_with_1 value being a better fit for the other class (i.e a high rows_with_1 in a cherry would make the model predict non-living since it thinks it's a golfclub and vice versa).
In the next section, when we classify images based on their label, we will have to carefully consider such cases and potentially reconsider our choice of predictor variables.

```
methods <- c()
# Caclulate TP, FP, TN, FN
TP <- cm$table[2, 2]
FP <- cm$table[2, 1]
TN <- cm$table[1, 1]
FN <- cm$table[1, 2]
# calculate the difference performance metrics and store in
# a list
accuracy <- model$results$Accuracy
methods <- append(methods, accuracy)
true_positive <- TP/(TP + FN)
methods <- append(methods, true_positive)
```

```
false_positive <- FP/(FP + TN)
methods <- append(methods, false_positive)
precision <- TP/(TP + FP)
methods <- append(methods, precision)
F1 <- 2 * ((precision * true_positive)/(precision + true_positive))
methods <- append(methods, F1)

# store all performance metrics in a df
results <- data.frame(c("Accuracy", "True Postive Rate/Recall",
    "False Positive Rate", "Precision", "F1 Score"), methods)

kable(results, col.names = c("Evaluation Method", "Value"), caption = "A table showing the value of dif
```

Table 7: A table showing the value of different methods of evaluating our model:

Evaluation Method	Value
Accuracy	0.9464286
True Postive Rate/Recall	0.9285714
False Positive Rate	0.0357143
Precision	0.9629630
F1 Score	0.9454545

The model achieved an accuracy of 94.6% using 7-fold cross validation, indicating again that the selected predictor variables are a good choice.

The true positive rate, also called recall or sensitivity, for this model is: $\frac{\text{TP}}{\text{TP + FN}} = \frac{52}{52+4} = 0.9285$. Our model predicted 4 living things (1 bananas and 3 cherries) as non-living things. Meaning 4 of the 6 mistakes were false negatives and therefore our recall will attain lowest value out of accuracy, itself, precision and F1 score.

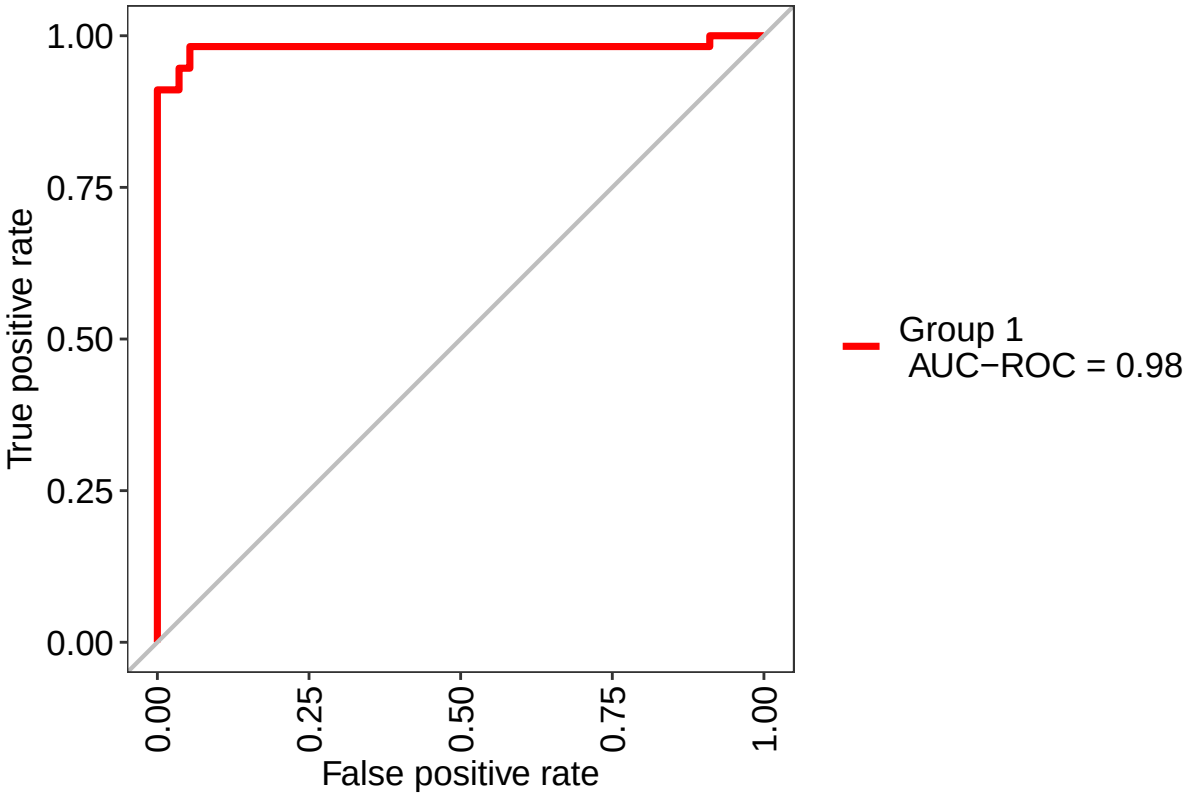
The false positive rate, for this model is: $\frac{\text{FP}}{\text{FP + TN}} = \frac{2}{2+54} = 0.0357$. As the model only misclassified two golfclubs as living things, resulting in only two false positives, we have a very low and significant false positive rate here.

The precision of this model is: $\frac{\text{TP}}{\text{TP + FP}} = \frac{52}{52+2} = 0.96$, which means that out of all the items predicted as living things, 96% of them are actually living things. This high precision value is not surprising given that we only had two false positives in our predictions.

F1 Score is described as: $2 \times \frac{\text{precision} \times \text{recall}}{\text{precision + recall}} = 0.945$. Similarly to the previous model, the high F1 Score is indicative of a high precision and recall and is an important evaluation metric for the model.

Section 1.3

```
library(MLeval)
# get the roc plot only (i.e showplot=FALSE, silent=TRUE)
res <- evalm(model, showplots = FALSE, silent = TRUE)
res$roc
```



The ROC curve, short for receiver operating characteristic curve, is a graphical representation of a classification model's performance across all decision thresholds. The perfect ROC curve has an AUC (Area Under Curve) value of 1. Meaning that it classified every item in the data set correctly. In this case the curve would be comprised of a vertical line from (0,0) to (0,1) and a horizontal line from (0,1) to (1,1). In our case, the ROC curve roughly takes this shape, indicating that the model is performing very well. The AUC value of 0.98 further confirms this notion.

Based on our analysis, we have identified four features that can be effectively used together in a logistic regression model to accurately predict the category (living or non-living) of any of our 112 images. However, to predict the specific label of each image, we need to use a different model. In the next section, we will explore a KNN (k-nearest neighbours) model as an alternative to binomial logistic regression.

Section 2

In this section we will perform 8-way classification for each of the object classes. We will build a model using KNN classification and analyse the accuracy we have over the 112 images.

Section 2.1

In this section we will develop a k-nearest-neighbour classification model for all odd values of k between 1-15 (inclusive). First, we must determine the optimal combination of four features that will most accurately predict the label of any given image. We will evaluate the accuracy of the model corresponding each value of k over the full set of 112 items.

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
library(class)

# remove living thing feature from df
csv_df <- subset(csv_df, select = -c(livingThing))
```

```
# standardise data:
standardized.X <- scale(csv_df[, c(12, 15, 17, 18)])  #rows_with_1, aspect_ratio, hollowness, custom

data.X <- standardized.X  #values
data.Y <- csv_df$labels  #labels
```

Note: To carry out further analysis we need to scale our data, as we did above, since our features are not on the same scale (e.g eyes take values of 0,1,2,3 and nr pix all have values > 60).

It is worth noting that for this section, we have selected different variables than in the first section to predict the label of any given image. In the previous section, we were only predicting whether an image belonged to living or non-living things, and therefore, the features we used do not necessarily give us any information between specific image labels.

To select the predictor variables, we first kept the best predictor feature for distinguishing between living and non-living things from the previous assignment (custom), which narrowed down the image to a subset of four image classes. We then selected three variables that best narrowed down each of those subsets to one particular image class.

For example, a low curvature combined with a hollowness of 0 (-1.12293571 after scaling) indicated a non-living image with no eyes, which leaves only the wine glass. Similarly, a higher curvature and no eyes indicated a tree. From here, we needed to find variables that helped us when we did not have such concrete cases. For example, a non-living thing with a hollowness of around 1 (-0.455 after scaling) could either be a golf club or a pencil, making it challenging for our model to predict which image class it belongs to. However, using connected areas as our next predictor helped differentiate between golf clubs, which generally have three connected areas (2.4548258 after scaling), and pencils, which generally have one connected area (-0.5336578 after scaling).

With these three features, we should be able to achieve close to 100% accuracy on trees, wine glasses, golf clubs, pencils, and envelopes. We then selected aspect ratio as the final feature, since in general bananas, cherries, and lemons have differentiating values for this feature.

Now let us evaluate the performance and accuracy of our model for all odd values of k between 1 and 15.

```
# create a prediction model for every odd value between 1
# and 15
knn.pred1 <- knn(data.X, data.X, data.Y, k = 1)
knn.pred3 <- knn(data.X, data.X, data.Y, k = 3)
knn.pred5 <- knn(data.X, data.X, data.Y, k = 5)
knn.pred7 <- knn(data.X, data.X, data.Y, k = 7)
knn.pred9 <- knn(data.X, data.X, data.Y, k = 9)
knn.pred11 <- knn(data.X, data.X, data.Y, k = 11)
knn.pred13 <- knn(data.X, data.X, data.Y, k = 13)
knn.pred15 <- knn(data.X, data.X, data.Y, k = 15)

# what proportion did we get wrong in each case?
acc <- c()
acc <- append(acc, 1 - mean(data.Y != knn.pred1))
acc <- append(acc, 1 - mean(data.Y != knn.pred3))
acc <- append(acc, 1 - mean(data.Y != knn.pred5))
acc <- append(acc, 1 - mean(data.Y != knn.pred7))
acc <- append(acc, 1 - mean(data.Y != knn.pred9))
acc <- append(acc, 1 - mean(data.Y != knn.pred11))
acc <- append(acc, 1 - mean(data.Y != knn.pred13))
acc <- append(acc, 1 - mean(data.Y != knn.pred15))
```

```
ks <- c()
ks <- ((1:8) * 2) - 1

df <- data.frame(ks, acc)

kable(df, col.names = c("K", "Accuracy"), caption = "A table showing the accuracy for each value of k:")
```

Table 8: A table showing the accuracy for each value of k:

K	Accuracy
1	1.0000000
3	0.9375000
5	0.9464286
7	0.9285714
9	0.9285714
11	0.9107143
13	0.9107143
15	0.9196429

The accuracy of the model is considerably high for all values of k, as depicted in the table above. It is important to note however, that a 100% accuracy score for k=1 is not an impressive or useful result since we are making predictions on data that the model has already been trained on. In fact, a higher value of k is expected to result in a lower accuracy score. The table above confirms this as we can see that there is a negative correlation between the accuracy and value of higher values of k. So, while our k=1 model appears 'perfect', if we gave it an unseen test set to predict it would likely preform poorly since it is overfit to the training data.

This is an unnatural way to train a model and to address this issue, we generally implement either seperate training and test sets or cross-validation techniques. In the upcoming subsection, we will use 7-fold cross-validation to re-evaluate the accuracy of our model for various values of k.

Section 2.2

In this section we create another classification model using the same 4 features and values of k as above, however, we will use 7-fold cross validation to avoid overfitting to the training data.

```
# 2.2
kfoldsk = 7
# train 7-fold cross validated model
train_control <- trainControl(method = "cv", number = kfoldsk,
    savePredictions = TRUE, classProbs = TRUE)

# the values of k we want to use in our models
tune_grid <- expand.grid(k = ((1:8) * 2) - 1)

# train the model on the entire dataset
model <- train(labels ~ connected_areas + aspect_ratio + hollowness +
    custom, data = csv_df, trControl = train_control, preProcess = c("center",
    "scale"), tuneGrid = tune_grid, method = "knn")

# summarize results
print(model)
```

```
## k-Nearest Neighbors
```

```
##
## 112 samples
##   4 predictor
##   8 classes: 'banana', 'cherry', 'envelope', 'golfclub', 'lemon', 'pencil', 'tree', 'wineglass'
##
## Pre-processing: centered (4), scaled (4)
## Resampling: Cross-Validated (7 fold)
## Summary of sample sizes: 96, 96, 96, 96, 96, 96, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.8928571  0.8775510
##    3  0.9196429  0.9081633
##    5  0.9285714  0.9183673
##    7  0.8928571  0.8775510
##    9  0.8839286  0.8673469
##   11  0.9107143  0.8979592
##   13  0.9017857  0.8877551
##   15  0.9107143  0.8979592
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

The best value of k was found to be 5 with a cross-validation accuracy of approximately 93% and a kappa value of 0.92, indicating that the model performs significantly better than a randomly created model. Lower values of k resulted in lower accuracy due to overfitting, while higher values of k also result in lower accuracy due to underfitting as expected.

Now let us investigate the predictions made by the model on the entire 112 items.

Section 2.3

```
# do a prediction on the entire dataset
knnPredict <- predict(model, newdata = csv_df)
# confusion matrix of results
cm = confusionMatrix(table(knnPredict, csv_df$labels))
kable(cm$table, caption = "Confusion Matrix of KNN model with K=5 over the 112 images:")
```

Table 9: Confusion Matrix of KNN model with K=5 over the 112 images:

	banana	cherry	envelope	golfclub	lemon	pencil	tree	wineglass
banana	13	0	0	0	0	0	0	0
cherry	1	14	0	0	0	0	0	0
envelope	0	0	14	0	0	0	0	0
golfclub	0	0	0	14	0	0	0	0
lemon	0	0	0	0	14	0	0	0
pencil	0	0	0	0	0	14	0	0
tree	0	0	0	0	0	0	10	1
wineglass	0	0	0	0	0	0	4	13

```
mean(knnPredict == csv_df$labels)
```

```
## [1] 0.9464286
```

The model achieved an impressive accuracy of 94.6% when tested on the entire set of 112 images, with only 6 incorrect predictions. It's worth noting that the overall accuracy of our model, which is calculated over the entire dataset, is higher than the 7-fold cross validated accuracy we obtained earlier. This is not surprising and a common occurrence since we are predicting items in the training set and not an unseen test set.

However, the results showed that the model struggled with distinguishing between trees and wineglasses, which was unexpected given our rationale of feature selection. Recall we previously claimed the combination of curvature and hollowness would be able to distinguish trees and wineglasses. This assumption was made based on the idea that curvature is the most prominent feature for differentiating between living and non-living objects, and hollowness would indicate zero eyes in both cases. This assumption proved to be naive, as in many cases, trees and wineglasses would of course have similar curvature!

In hindsight, it would have been better to select an additional distinguishing feature besides hollowness and curvature. Rows_with_1 comes to mind, as trees generally do not have any rows_with_1, while the handle of a wineglass typically has a high value of rows_with_1.

Out of interest, let us remove aspect_ratio from our predictor variables and replace it with rows_with_1 and re-evaluate our models performance.

```
# alternative model
model2 <- train(labels ~ connected_areas + rows_with_1 + hollowness +
    custom, data = csv_df, trControl = train_control, preProcess = c("center",
    "scale"), tuneGrid = tune_grid, method = "knn")
# get its predictions and confusion matrix and compare
# results
knnPredict2 <- predict(model2, newdata = csv_df)
cm2 = confusionMatrix(table(knnPredict2, csv_df$labels))
kable(cm2$table, caption = "Confusion Matrix of our adjusted KNN model over the 112 images:")
```

Table 10: Confusion Matrix of our adjusted KNN model over the 112 images:

	banana	cherry	envelope	golfclub	lemon	pencil	tree	wineglass
banana	14	0	0	0	0	0	0	0
cherry	0	14	0	0	0	0	0	0
envelope	0	0	14	0	0	0	0	0
golfclub	0	0	0	14	0	0	0	0
lemon	0	0	0	0	14	0	0	0
pencil	0	0	0	0	0	14	0	0
tree	0	0	0	0	0	0	14	0
wineglass	0	0	0	0	0	0	0	14

```
mean(knnPredict2 == csv_df$labels)
```

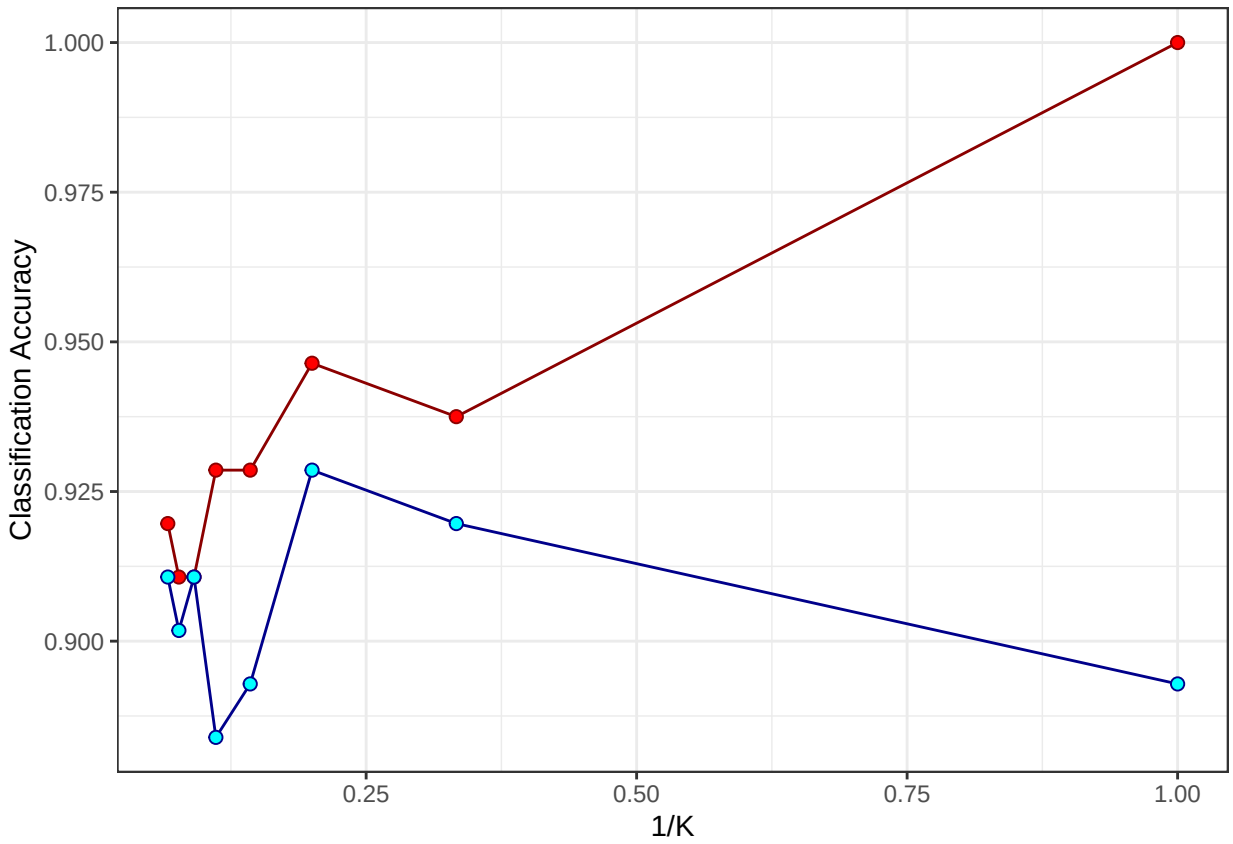
```
## [1] 1
```

Our model is 100% accurate now! This emphasizes that the importance of selecting appropriate predictor variables cannot be underestimated. We see now that our initial assumption that the combination of curvature and hollowness would be sufficient for distinguishing between trees and wineglasses was not accurate. We will continue the rest of our analysis with the original 4 predictor variables we used but it is worth noting here that often times when we are creating prediction models it can be difficult for us to intuitively choose the optimal features and it requires some more in depth analysis (as we have done above) to find the optimal model.

Section 2.4

In this section we will plot the original model and the cross validated model against each other and compare the trends for each value of k.

```
library(ggplot2)

ks <- c()
ks <- 1/(((1:8) * 2) - 1)
# extract accuracy
knn_df <- data.frame(acc, ks, model$results[, 2])

# create plot
plot1 <- ggplot(knn_df, aes(x = ks)) + geom_line(aes(y = acc),
    color = "darkred") + geom_point(aes(y = acc), shape = 21,
    color = "darkred", fill = "red", size = 2) + geom_line(aes(y = model$results[,
    2]), color = "darkblue") + geom_point(aes(y = model$results[,
    2]), shape = 21, color = "darkblue", fill = "cyan", size = 2) +
    labs(x = "1/K", y = "Classification Accuracy") + theme_bw()

plot1
```



In the above graph the red line represents our original model when we were over fitting to the training data and the blue line represents our cross validated model.It's important to note that the x-axis represents the inverse of the number of neighbors (1/k). We do this to constrain our range to fall between 0 and 1. As a result, a high value of 1/k corresponds to a low value of k, and conversely, a low value of 1/k corresponds to a high value of k.

As expected our original model takes an upward trend meaning that it has a higher accuracy for lower values

of k. This intuitively makes sense since our model has 100% accuracy when we use k=1 because our training data is the same as our test data.

In contrast, the cross validated model shows a peak in accuracy around 1/k = 0.2, with lower accuracies for the more extreme values of 1/k at either end of the graph. This trend makes sense as we aim to avoid overfitting and underfitting to the training data. The model's highest accuracy was at k=5, so a peak at 1/k = 0.2 is expected.

Section 3

In this section we are using a much larger data set of 800 training items (100 from each image class) and a test data set of 400 items (50 from each image class). The features remain the same, with the exception of the custom feature now being the number of black pixels with exactly one black pixel neighbour. The values in each case have been updated to match the new data set.
We will preform classification using random forests with respect to the 8 image categories using all the features from our large dataset.

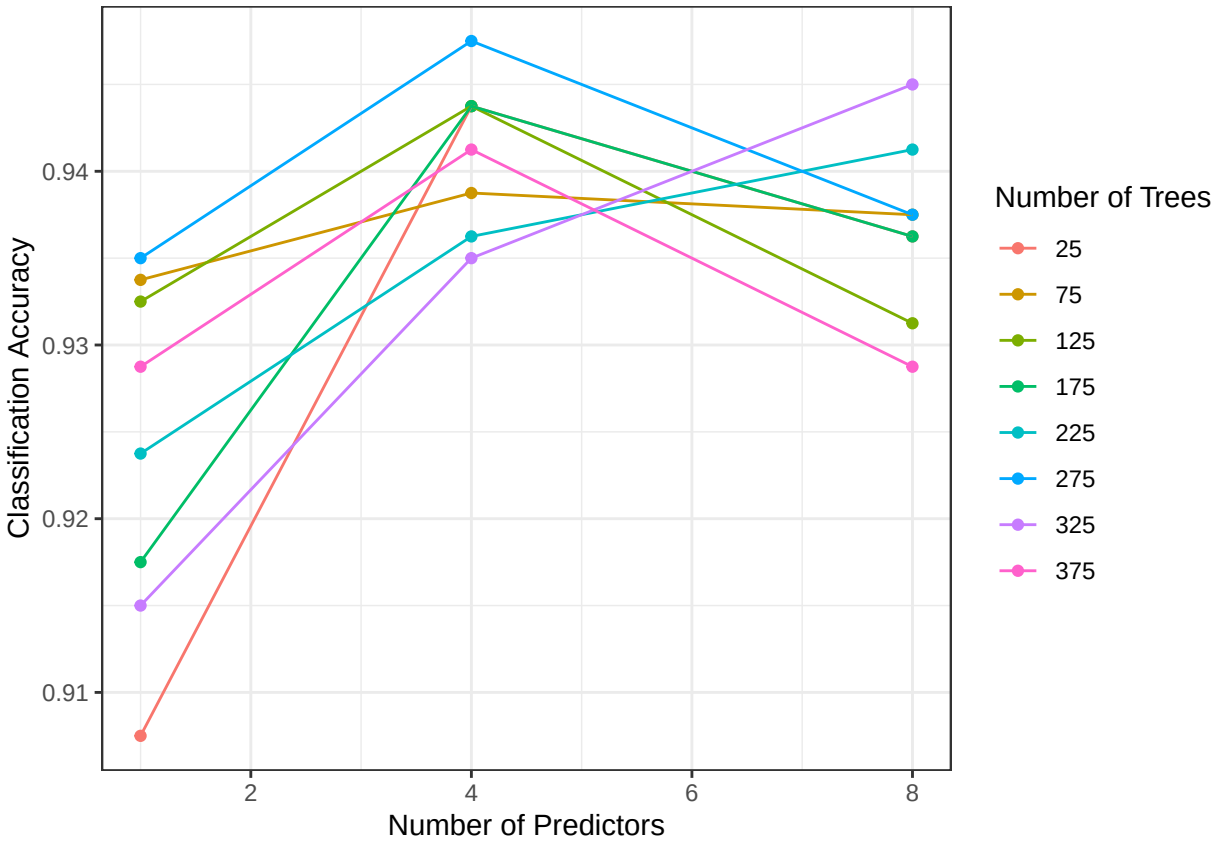
Section 3.1

```
# Set the path to the folder containing the training files
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd(file.path(getwd(), "/a3_data/traindata/30leyp9ykm"))

# convert all features file into a dataframe
all_features <- read.delim("all_features.csv", header = FALSE,
    stringsAsFactors = FALSE, quote = "", sep = "\t")
colnames(all_features) <- c("labels", "indexs", "nr_pix", "rows_with_1",
    "cols_with_1", "rows_with_2", "cols_with_2", "rows_with_3p",
    "cols_with_3p", "height", "width", "aspect_ratio", "maxrow",
    "maxcol", "connected_areas", "eyes", "hollowness", "custom")

# Set the path to the folder containing the testdata files
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
setwd(file.path(getwd(), "/a3_data/testdata"))

# Get a list of all the files in the folder
test_data <- list.files(path = getwd())
test_features <- list()

# filter out to just get the features data file of each
# image and combine them into one dataframe
for (test_file in test_data) {
    if (str_detect(test_file, "features.csv")) {
        test_feature <- read.delim(test_file, header = FALSE,
            stringsAsFactors = FALSE, quote = "", sep = "\t")
        test_features <- rbind(test_features, test_feature)
    }
}
colnames(test_features) <- c("labels", "indexs", "nr_pix", "rows_with_1",
    "cols_with_1", "rows_with_2", "cols_with_2", "rows_with_3p",
    "cols_with_3p", "height", "width", "aspect_ratio", "maxrow",
    "maxcol", "connected_areas", "eyes", "hollowness", "custom")
```

We will now perform classification with random forests using 5-fold cross-validation. We will be calculating

multiple random forest solutions using number of trees, Nt, between 25 and 375 (increments of 50) and number of predictors, considered for each Nt, is defined by Np = {1, 4, 8}. The end goal is to find the combination of tree-number (Nt) and predictor-number (Np) yielding the best cross-validated accuracy.

```
# 3.1
library("randomForest")

# change labels from chr type to factor
all_features$labels <- as.factor(all_features$labels)
test_features$labels <- as.factor(test_features$labels)

# training data
training <- all_features

# testing data
testing <- test_features

kfoldsk = 5  #train our model using 5 fold cv
train_control <- trainControl(method = "cv", number = kfoldsk,
    savePredictions = TRUE, classProbs = TRUE)

# values of mtry our model should use
tune_grid <- expand.grid(mtry = c(1, 4, 8))

num_trees <- c(25, 75, 125, 175, 225, 275, 325, 375)  #25<=Nt<=375
rfm_results <- list()
rfm_models <- list()

# loop through all our values for Nt (Np is done via
# tunegrid) and create a model each time
for (nt in num_trees) {
    RFM <- train(labels ~ ., data = training, ntree = nt, trControl = train_control,
        preProcess = c("center", "scale"), tuneGrid = tune_grid,
        method = "rf")
    rfm_results <- rbind(rfm_results, RFM$results)  #store each models result in list
    rfm_models <- rbind(rfm_models, RFM$finalModel)  #store each model in list

}
rfm_results <- cbind(rfm_results, num_trees)
# tidying up the ordering of rows and columns
rfm_results <- rfm_results[, c(6, 1, 2, 3, 4, 5)]
rfm_results <- rfm_results[c(1, 17, 9, 10, 2, 18, 19, 11, 3,
    4, 20, 12, 13, 5, 21, 22, 14, 6, 7, 23, 15, 16, 8, 24), ]
rownames(rfm_results) <- c("1":"24")
colnames(rfm_results) <- c("Nt", "Np", "Accuracy", "Kappa", "AccuracySD",
    "KappaSD")

# create a plot of all our 24 models accuracies against Np
rf_plot <- ggplot(rfm_results, aes(Np, Accuracy, colour = factor(Nt))) +
    geom_point() + geom_line() + labs(x = "Number of Predictors",
    y = "Classification Accuracy", colour = "Number of Trees") +
    theme_bw()
rf_plot
```



The figure above presents a clear visualization of our 24 different random forest models. It is not surprising that the worst performing model has Nt = 25 and Np = 1 as this model is clearly underfitting the data. We observe a general trend that the best choice of Np tends to be 4 for each Nt. This aligns with the default approach of choosing the number of predictors, which is to take the square root of the total number of predictors for classification. In our case: $\sqrt{\text{number of predictors}} = \sqrt{16} = 4$. For six of our eight choices of Nt we see this is in fact the best choice of Np. Letting Np be 8 works better for two of the models but these appear to be slight outliers. In the other cases using 8 predictors might be subject to overfitting or make our ensemble vote more correlated which we want to avoid. In general, for random forests, a higher number of trees increases the performance and makes the predictions more stable. We can see this is mostly true for our models. With all this in mind it should not come as a surprise that our most accurate model takes Nt = 275, Np = 4 and it yields a cross validation accuracy of 94.75%.

```
# extract the model with the highest cross validated
# accuracy
best_model <- rfm_results[which.max(rfm_results$Accuracy), ]
best_model
```

```
##     Nt Np Accuracy Kappa AccuracySD    KappaSD
## 17 275  4   0.9475  0.94 0.01568738 0.01792843
```

```
# run the best model again
tune_grid <- expand.grid(mtry = c(4))
RFM <- train(labels ~ ., data = training, ntree = 275, trControl = train_control,
    preProcess = c("center", "scale"), tuneGrid = tune_grid,
    method = "rf")

# confusion matrix of the best model
cm <- confusionMatrix(table(RFM$pred$pred, RFM$pred$obs))
# evaluation of the best models performance
```

```
eval_table <- cm$byClass[, c("Sensitivity", "Specificity", "Precision",
    "F1", "Balanced Accuracy")]

names(dimnames(cm$table)) <- c("Predicted", "Actual")
kable(cm$table, caption = "Confusion Matrix of our model on the test dataset")
```

Table 11: Confusion Matrix of our model on the test dataset

	banana	cherry	envelope	golfclub	lemon	pencil	tree	wineglass
banana	87	3	0	3	2	1	2	1
cherry	0	92	0	0	1	0	3	1
envelope	0	0	100	0	0	0	2	0
golfclub	3	0	0	97	0	0	0	0
lemon	1	0	0	0	95	0	1	0
pencil	8	4	0	0	1	99	0	0
tree	1	1	0	0	1	0	91	3
wineglass	0	0	0	0	0	0	1	95

The confusion matrix above reveals that the model struggles in accurately classifying bananas. It failed to classify 13 bananas correctly, classifying 8 of them as pencils. This could be attributed to the presence of outliers in certain banana features. For instance, if a banana has 2 eyes, generally speaking, its other feature values significantly overlap with those of a pencil, making it more challenging for the model to classify it correctly. In fact consider the following:

```
# find the outliers of bananas and normal cases for pencil
# and print result
outlier_banana_eyes <- all_features[(all_features$eyes == 2 |
    all_features$eyes == 3), ]
outlier_banana_eyes <- outlier_banana_eyes[outlier_banana_eyes$labels ==
    "banana", ]
normal_pencil_eyes <- all_features[(all_features$eyes == 2 |
    all_features$eyes == 3), ]
normal_pencil_eyes <- normal_pencil_eyes[normal_pencil_eyes$labels ==
    "pencil", ]

cat("Number of Bananas with 2/3 eyes:", nrow(outlier_banana_eyes),
    "       Number of Pencils with 2/3 eyes:", nrow(normal_pencil_eyes))
```

```
## Number of Bananas with 2/3 eyes: 8        Number of Pencils with 2/3 eyes: 97
```

Exactly 8 of the bananas in our training dataset have 2/3 eyes while 97 of the pencils have 2/3 eyes. Hence we can boldly assume the 8 cases when our model classified a banana as a pencil was caused by this. Although likely, this is not guaranteed to be the case due to the randomness of the feature selection in individual trees in random forests, we may not be using eyes to make predictions at the optimal node in majority of the models. This means it could be other features causing the populous vote to be pencil. Conversely, since 97 of the pencils had 2/3 eyes we wouldn't expect many of them to be wrongly classified as bananas and we can see this only occurred once in our model.

Other noticeable images classes that had trouble being classified correctly were trees and cherries. Interestingly the only class entirely correctly classified was the envelope class.

Let us now formally evaluate our model:

```
kable(eval_table, caption = "Evaluation of the Best Models performance:")
```

Table 12: Evaluation of the Best Models performance:

	Sensitivity	Specificity	Precision	F1	Balanced Accuracy
Class: banana	0.87	0.9828571	0.8787879	0.8743719	0.9264286
Class: cherry	0.92	0.9928571	0.9484536	0.9340102	0.9564286
Class: envelope	1.00	0.9971429	0.9803922	0.9900990	0.9985714
Class: golfclub	0.97	0.9957143	0.9700000	0.9700000	0.9828571
Class: lemon	0.95	0.9971429	0.9793814	0.9644670	0.9735714
Class: pencil	0.99	0.9814286	0.8839286	0.9339623	0.9857143
Class: tree	0.91	0.9914286	0.9381443	0.9238579	0.9507143
Class: wineglass	0.95	0.9985714	0.9895833	0.9693878	0.9742857

```
cat("Overall cross validated accuracy:", sum(diag(cm$table))/800)
```

```
## Overall cross validated accuracy: 0.945
```

As expected, the envelopes were found to have the highest sensitivity and in fact achieved the best performance across all metrics. However, despite achieving perfect sensitivity, the envelopes did not have 100% specificity or precision since the model mistakenly classified 2 trees as envelopes (i.e false positives). As for the banana class, it exhibited the poorest performance across all metrics, which was expected based on the results of the confusion matrix. Overall, the model achieved a cross-validated accuracy of 94.5%.

Section 3.2

Random forests and cross-validation both involve a certain level of randomness, which is one of their main advantages compared to basic decision trees or bagging algorithms. This randomness is helpful because it means we're not always using the best prediction feature to build each branch - instead, we're using a random subset of features that may not include the best feature. This approach reduces variance, decorrelates the ensemble voting, and mitigates the effects of the greedy algorithm used in building trees.

However, the accuracy of the model can vary across different independent runs due to this randomness. In this section, we will assess the variability in accuracy across different independent runs. Using our best model in the above section we will refit the model 12 times, to obtain 12 cross-validated accuracy scores. Our main goal is to check if the model preforms significantly better than chance. Since we have a small sample size (n) of 12, assuming a normal distribution in our data, a suitable test for us to conduct will be a t-test.

```
refitting_results <- list()
# create 12 different models and store the results of each
# in a list
for (i in seq_along(1:12)) {
    RFM <- train(labels ~ ., data = training, ntree = 275, trControl = train_control,
        preProcess = c("center", "scale"), tuneGrid = tune_grid,
        method = "rf")
    refitting_results <- rbind(refitting_results, RFM$results)
}
# create table of accuracy and kappa values for each model
table_df <- cbind(refitting_results$Accuracy, refitting_results$Kappa)
colnames(table_df) <- c("Accuracy", "Kappa")
rownames(table_df) <- c("1":"12")
kable(table_df, row.names = TRUE, caption = "Accuracy for each Refit of our Best Model")
```

Table 13: Accuracy for each Refit of our Best Model

	Accuracy	Kappa
1	0.93250	0.9228571
2	0.94250	0.9342857
3	0.94250	0.9342857
4	0.94125	0.9328571
5	0.93875	0.9300000
6	0.93625	0.9271429
7	0.94750	0.9400000
8	0.94125	0.9328571
9	0.94750	0.9400000
10	0.94250	0.9342857
11	0.94250	0.9342857
12	0.94875	0.9414286

Seen above is the resultant accuracies of each run of our model. Now we will calculate the mean (point estimate), standard deviation, standard error and test statistic of our models accuracy across the 12 runs. Recall the standard error and test statistic are calculated as follows:
Standard Error $= \frac{\text{Standard Deviation}}{\sqrt{n}}$, $T_{df} = \frac{\text{Point Estimate} - \text{Null Value}}{\text{Standard Error}}$

```
# find the point estimate, SD, SE and t-statistic
point_estimate <- mean(refitting_results$Accuracy)
SD <- sd(refitting_results$Accuracy)
SE <- SD/sqrt(12)
null_value <- 1/8
T_11 <- (point_estimate - null_value)/SE

# create a table to show each
t_stats <- cbind(point_estimate, SD, SE, null_value, T_11)
colnames(t_stats) <- c("Point Estimate", "SD", "SE", "Null Value",
    "Test Statistic")

kable(t_stats, caption = "Statistical Features of our Model")
```

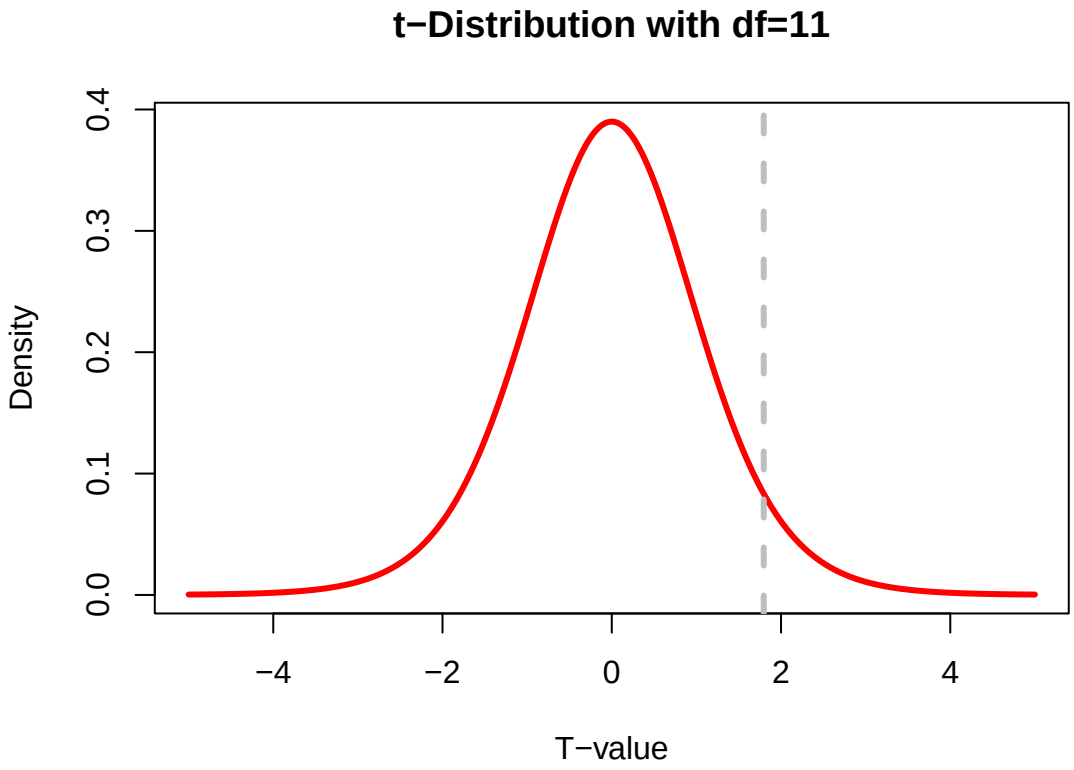
Table 14: Statistical Features of our Model

Point Estimate	SD	SE	Null Value	Test Statistic
0.9419792	0.004691	0.0013542	0.125	603.3077

To test if our model's performance is better than chance, we set our null value to 0.125, which represents the accuracy we would expect if the model were predicting 8 classes purely by chance. Notice that our sample mean of 0.94 is much higher than this value, indicating that the model is performing well. The test statistic yields a large value of 600, and the standard error is extremely small at 0.0046. With these results, we can confidently expect that our model is significantly better than the model of chance. To further examine this, we will plot the t-distribution for a sample of size 12 (i.e., df = 11)

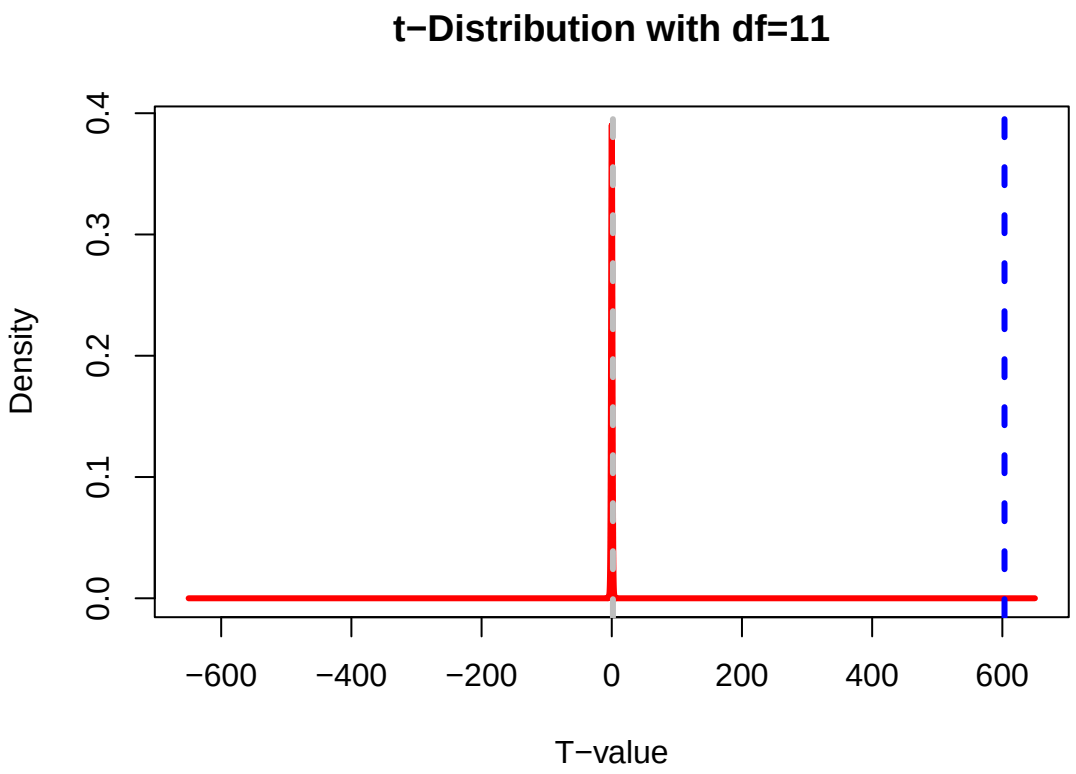
```
x <- -500:500/100  # The range we are going to plot
hx <- dt(x, 11)  # Probability density for each x
rtail = qt(0.95, 11)  # top 5%

# plot t-dist
```

```
plot(x, hx, type = "l", col = "red", lty = 1, lwd = 3, xlab = "T-value",
    ylab = "Density", main = "t-Distribution with df=11")
abline(v = rtail, col = "grey", lty = 2, lwd = 3)
```



The grey line here represents the 95% cut off and any value beyond this point would be considered 'significantly better' than the model based on chance. To show where our value falls we will need a much large range...

```
# plotting T-dist
x <- -65000:65000/100  # The range we are going to plot
hx <- dt(x, 11)  # Probability density for each x
rtail = qt(0.95, 11)  # top 5%
plot(x, hx, type = "l", col = "red", lty = 1, lwd = 3, xlab = "T-value",
    ylab = "Density", main = "t-Distribution with df=11")
abline(v = T_11, col = "blue", lty = 2, lwd = 3)
abline(v = rtail, col = "grey", lty = 2, lwd = 3)
```



The results show that our model is an extreme example of a model preforming significantly better than the model based purely on chance. This is not surprising given the effort we put into creating the model and selecting appropriate prediction features. We can see the p-value in t-test call below reinforces our confidence in the model's performance:

```
# preform t-test
t.test(refitting_results$Accuracy, alternative = "greater", mu = 0.125,
    conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  refitting_results$Accuracy
## t = 603.31, df = 11, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 0.125
## 95 percent confidence interval:
##  0.9395472       Inf
## sample estimates:
## mean of x
## 0.9419792
```

Section 3.3

In the final section we will perform an evaluation of our best model's performance on the test set of 400 unseen images.

```
# evaluating the model
labels_pred <- predict(RFM, testing)
testing$labels_pred <- labels_pred

# confusion matrix
```

```
cm <- confusionMatrix(table(testing$labels_pred, testing$labels))
kable(cm$table, caption = "Confusion Matrix Predicting the Test Data")
```

Table 15: Confusion Matrix Predicting the Test Data

	banana	cherry	envelope	golfclub	lemon	pencil	tree	wineglass
banana	42	3	0	0	2	2	0	0
cherry	1	46	0	0	1	0	1	0
envelope	0	0	50	0	0	0	2	0
golfclub	0	0	0	47	0	0	0	0
lemon	0	0	0	1	47	0	0	0
pencil	7	0	0	2	0	48	0	0
tree	0	0	0	0	0	0	46	2
wineglass	0	1	0	0	0	0	1	48

The model's performance on the test data followed a similar trend to its performance during cross-validation predictions, struggling to accurately classify the banana class. In fact, in 7 out of 8 cases where the model misclassified bananas, it classified them as pencils. This outcome was anticipated based on the model's predictions on the training data. It is worth noting that the model once again correctly classified all envelopes.

Let us formally evaluate the models performance on the test set:

```
# extract the confusion matrix of our model and check its
# accuracy

eval_table <- cm$byClass[, c("Sensitivity", "Specificity", "Precision",
    "F1", "Balanced Accuracy")]
kable(eval_table, caption = "Evaluation of the Best Models performance:")
```

Table 16: Evaluation of the Best Models performance:

	Sensitivity	Specificity	Precision	F1	Balanced Accuracy
Class: banana	0.84	0.9800000	0.8571429	0.8484848	0.9100000
Class: cherry	0.92	0.9914286	0.9387755	0.9292929	0.9557143
Class: envelope	1.00	0.9942857	0.9615385	0.9803922	0.9971429
Class: golfclub	0.94	1.0000000	1.0000000	0.9690722	0.9700000
Class: lemon	0.94	0.9971429	0.9791667	0.9591837	0.9685714
Class: pencil	0.96	0.9742857	0.8421053	0.8971963	0.9671429
Class: tree	0.92	0.9942857	0.9583333	0.9387755	0.9571429
Class: wineglass	0.96	0.9942857	0.9600000	0.9600000	0.9771429

```
cat("Overall accuracy:", sum(diag(cm$table))/400)
```

```
## Overall accuracy: 0.935
```

As previously mentioned, the model's performance on the test set was consistent with its performance during cross-validation on the training data. As a result, it is not surprising that the envelopes once again achieved 100% sensitivity and scored the highest across all metrics, with 0 false negatives and 2 false positives. The banana class remained the worst performing class across all metrics, with the model incorrectly classifying 7 bananas as pencils. The overall accuracy for the test set was slightly lower at 93.5% compared to the cross-validation accuracy. This is a common trend where validation accuracy tends to be higher than test accuracy because the model's hyperparameters are typically tuned specifically for the validation dataset.

Therefore, the slightly lower accuracy is not unexpected or concerning and our model appears to be performing relatively well.